Immediate-Mode Reference

This section contains reference information for the API elements provided by Direct3D® Immediate Mode. Reference material is divided into the following categories:

- **Interfaces**
- **D3D_OVERLOADS**
- **Callback Functions**
- **Macros**
- **Structures**
- **Enumerated Types**
- **Other Types**
- **Return Values**

Interfaces

This section contains reference information for the COM interfaces provided by Direct3D's Immediate Mode. The following interfaces are covered:

- **IDirect3D2**
- **IDirect3DDevice**
- **IDirect3DDevice2**
- **IDirect3DExecuteBuffer**
- **IDirect3DLight**
- **IDirect3DMaterial2**
- **IDirect3DTexture2**
- **IDirect3DViewport2**

IDirect3D2

Applications use the methods of the **IDirect3D2** interface to create Direct3D objects and set up the environment. This section is a reference to the methods of this interface. For a conceptual overview, see IDirect3D2 interface.

The **IDirect3D2** interface is obtained by calling the **QueryInterface** method from a DirectDraw object.

The major difference between **IDirect3D2** and the **IDirect3D** interface is the addition of the **CreateDevice** method.

The methods of the **IDirect3D2** interface can be organized into the following groups:

Creation	**CreateDevice**
	CreateLight
	CreateMaterial
	CreateViewport
Enumeration	**EnumDevices**
	FindDevice

The **IDirect3D2** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3D2** and **LPDIRECT3D** types are defined as pointers to the **IDirect3D2** and **IDirect3D** interfaces:

```
typedef struct IDirect3D    *LPDIRECT3D;
typedef struct IDirect3D2   *LPDIRECT3D2;
```

IDirect3D2::CreateDevice

The **IDirect3D2::CreateDevice** method creates a Direct3D device to be used with the DrawPrimitive methods.

```
HRESULT CreateDevice(
  REFCLSID rclsid,
  LPDIRECTDRAWSURFACE lpDDS,
  LPDIRECT3DDEVICE2 * lplpD3DDevice2
);
```

Parameters

rclsid
Class identifier for the new device. This can be IID_IDirect3DHALDevice, IID_IDirect3DMMXDevice, IID_IDirect3DRampDevice, or IID_IDirect3DRGBDevice.

lpDDS
Address of a DirectDraw surface that describes the new device.

lplpD3DDevice2
Address that points to the new **IDirect3DDevice2** interface when the method returns.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3D2** interface. In previous versions of Direct3D, devices could be created only by calling the **IDirectDrawSurface::QueryInterface** method; devices created in this manner can only be used with execute buffers.

When you call **IDirect3D2::CreateDevice**, you create a device object that is separate from a DirectDraw surface object. This device uses a DirectDraw surface as a rendering target.

IDirect3D2::CreateLight

The **IDirect3D2::CreateLight** method allocates a Direct3DLight object. This object can then be associated with a viewport by using the **IDirect3DViewport2::AddLight** method.

```
HRESULT CreateLight(
  LPDIRECT3DLIGHT* lplpDirect3DLight,
  IUnknown* pUnkOuter
);
```

Parameters

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 **IDirect3D2::CreateLight** method returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3D** interface.

IDirect3D2::CreateMaterial

The **IDirect3D2::CreateMaterial** method allocates a Direct3DMaterial2 object.

```
HRESULT CreateMaterial(
  LPDIRECT3DMATERIAL2* lplpDirect3DMaterial2,
  IUnknown* pUnkOuter
);
```

Parameters

lplpDirect3DMaterial2
: Address that will be filled with a pointer to an **IDirect3DMaterial2** interface if the call succeeds.

pUnkOuter
: This parameter is provided for future compatibility with COM aggregation features. Currently, however, the **IDirect3D2::CreateMaterial** method returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error. For a list of possible return codes, see Direct3D Immediate-Mode Return Values.

Remarks

In the **IDirect3D** interface, this method retrieved a pointer to an **IDirect3DMaterial** interface, not an **IDirect3DMaterial2** interface.

IDirect3D2::CreateViewport

The **IDirect3D2::CreateViewport** method creates a Direct3DViewport object. The viewport is associated with a Direct3DDevice object by using the **IDirect3DDevice2::AddViewport** method.

```
HRESULT CreateViewport(
  LPDIRECT3DVIEWPORT2* lplpD3DViewport2,
  IUnknown* pUnkOuter
);
```

Parameters

lplpD3DViewport
: Address that will be filled with a pointer to an **IDirect3DViewport2** interface if the call succeeds.

pUnkOuter
: This parameter is provided for future compatibility with COM aggregation features. Currently, however, the **IDirect3D2::CreateViewport** method returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

In the **IDirect3D** interface, this method retrieves a pointer to an **IDirect3DViewport** interface, not an **IDirect3DViewport2** interface.

IDirect3D2::EnumDevices

The **IDirect3D2::EnumDevices** method enumerates all Direct3D device drivers installed on the system.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

MMX devices are enumerated only by **IDirect3D2::EnumDevices**, not by its predecessor, **IDirect3D::EnumDevices**. If you use the **QueryInterface** method to create an **IDirect3D** interface from **IDirect3D2** before you enumerate the Direct3D drivers, the enumeration will behave like **IDirect3D::EnumDevices** — no MMX devices will be enumerated.

To use execute buffers with an MMX device, you must call the **IDirect3D2::CreateDevice** method to create an MMX **IDirect3DDevice2** interface and then use the **QueryInterface** method to create an **IDirect3DDevice** interface from **IDirect3DDevice2**.

IDirect3D2::FindDevice

The **IDirect3D2::FindDevice** method finds a device with specified characteristics and retrieves a description of it.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error. For a list of possible return codes, see Direct3D Immediate-Mode Return Values.

Remarks

This method is unchanged from its implementation in the **IDirect3D** interface.

IDirect3D::Initialize

The **IDirect3D2::Initialize** method is not implemented.

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

IDirect3DDevice

Applications use the methods of the **IDirect3DDevice** interface to retrieve and set the capabilities of Direct3D devices. This section is a reference to the methods of these interface. For a conceptual overview, see Devices.

The **IDirect3DDevice** interface supports applications that work with execute buffers. It has been extended by the **IDirect3DDevice2** interface, which supports the DrawPrimitive methods.

The Direct3DDevice object is obtained by calling the **QueryInterface** method from a DirectDrawSurface object that was created as a 3-D–capable surface.

The methods of the **IDirect3DDevice** interface can be organized into the following groups. Note that in some cases **IDirect3DDevice** methods are documented in the reference to the **IDirect3DDevice2** interface.

Execute buffers	**CreateExecuteBuffer**
	Execute
Information	**EnumTextureFormats**
	GetCaps
	GetDirect3D
	GetPickRecords
	GetStats
Matrices	**CreateMatrix**
	DeleteMatrix
	GetMatrix
	SetMatrix
Miscellaneous	**Initialize**
	Pick
	SwapTextureHandles
Scenes	**BeginScene**

	EndScene
Viewports	**AddViewport**
	DeleteViewport
	NextViewport

The **IDirect3DDevice** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DDEVICE** type is defined as a pointer to the **IDirect3DDevice** interface:

```
typedef struct IDirect3DDevice        *LPDIRECT3DDEVICE;
```

IDirect3DDevice::CreateExecuteBuffer

The **IDirect3DDevice::CreateExecuteBuffer** method allocates an execute buffer for a display list.

```
HRESULT CreateExecuteBuffer(
  LPD3DEXECUTEBUFFERDESC lpDesc,
  LPDIRECT3DEXECUTEBUFFER *lplpDirect3DExecuteBuffer,
  IUnknown *pUnkOuter
);
```

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

The display 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.

The **D3DEXECUTEBUFFERDESC** structure describes the execute buffer to be created. At a minimum, the application must specify the size required. If the application specifies D3DDEBCAPS_VIDEOMEMORY in the **dwCaps** 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

The **IDirect3DDevice::CreateMatrix** method creates a matrix.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error, such as DDERR_INVALIDPARAMS.

See Also

IDirect3DDevice::DeleteMatrix, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::DeleteMatrix

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

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

Parameters

d3dMatHandle
: Matrix handle to be deleted.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error, such as DDERR_INVALIDPARAMS.

See Also

IDirect3DDevice::CreateMatrix, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::Execute

The **IDirect3DDevice::Execute** method executes a buffer.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

D3DEXECUTEDATA, **D3DINSTRUCTION**, **IDirect3DExecuteBuffer::Validate**

IDirect3DDevice::GetMatrix

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

```
HRESULT GetMatrix(
  D3DMATRIXHANDLE D3DMatHandle,
  LPD3DMATRIX lpD3DMatrix
);
```

Parameters

D3DMatHandle
Handle to the matrix to be retrieved.

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error, such as DDERR_INVALIDPARAMS.

See Also

IDirect3DDevice::CreateMatrix, **IDirect3DDevice::DeleteMatrix**, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::GetPickRecords

The **IDirect3DDevice::GetPickRecords** method retrieves the pick records for a device.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error.

Remarks

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::Initialize

The **IDirect3DDevice::Initialize** method is not implemented.

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

IDirect3DDevice::Pick

The **IDirect3DDevice::Pick** method executes a buffer without performing any rendering, but returns a z-ordered list of offsets to the primitives that intersect the upper-left corner of the rectangle specified by *lpRect*.

This call fails if the Direct3DExecuteBuffer object is locked.

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

Parameters

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 device coordinates to be picked. Currently, only primitives that intersect the **x1, y1** coordinates of this rectangle are returned. The **x2, y2** coordinates are ignored.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

D3DERR_EXECUTE_LOCKED
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

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::SetMatrix

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

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

Parameters

d3dMatHandle
: Matrix handle to be set.

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error, such as DDERR_INVALIDPARAMS.

Remarks

Transformations inside the execute buffer include a handle to 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**

IDirect3DDevice2

The **IDirect3DDevice2** interface helps applications work with the DrawPrimitive methods; this is in contrast to the **IDirect3DDevice** interface, which applications use to work with execute buffers. You can create a Direct3DDevice2 object by calling the **IDirect3D2::CreateDevice** method.

For a conceptual overview, see Devices and The DrawPrimitive Methods.

The methods of the **IDirect3DDevice2** interface can be organized into the following groups:

Information	**EnumTextureFormats**
	GetCaps
	GetDirect3D
	GetStats
Miscellaneous	**MultiplyTransform**
	SwapTextureHandles
Getting and Setting States	**GetClipStatus**
	GetCurrentViewport
	GetLightState
	GetRenderState
	GetRenderTarget
	GetTransform
	SetClipStatus
	SetCurrentViewport
	SetLightState
	SetRenderState
	SetRenderTarget
	SetTransform
Rendering	**Begin**
	BeginIndexed
	DrawIndexedPrimitive
	DrawPrimitive
	End
	Index
	Vertex
Scenes	**BeginScene**
	EndScene
Viewports	**AddViewport**
	DeleteViewport
	NextViewport

The **IDirect3DDevice2** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **IDirect3DDevice2** interface is not intended to be used with execute buffers. If you need to use some of the methods in the **IDirect3DDevice** interface that are not supported in **IDirect3DDevice2**, you can call **IDirect3DDevice2::QueryInterface** to retrieve a pointer to an **IDirect3DDevice** interface. The following methods from the **IDirect3DDevice** interface are not supported by **IDirect3DDevice2**:

IDirect3DDevice::CreateExecuteBuffer

IDirect3DDevice::CreateMatrix

IDirect3DDevice::DeleteMatrix

IDirect3DDevice::Execute

IDirect3DDevice::GetMatrix

IDirect3DDevice::GetPickRecords

IDirect3DDevice::Initialize

IDirect3DDevice::Pick

IDirect3DDevice::SetMatrix

The **LPDIRECT3DDEVICE2** type is defined as a pointer to the **IDirect3DDevice2** interface:

```
typedef struct IDirect3DDevice2    *LPDIRECT3DDEVICE2;
```

IDirect3DDevice2::AddViewport

The **IDirect3DDevice2::AddViewport** method adds the specified viewport to the list of viewport objects associated with the device.

```
HRESULT AddViewport(
  LPDIRECT3DVIEWPORT2 lpDirect3DViewport2
);
```

Parameters

lpDirect3DViewport2
Address of the **IDirect3DViewport2** interface that should be associated with this Direct3DDevice object.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

In the **IDirect3DDevice** interface, this method requires a pointer to an **IDirect3DViewport** interface, not an **IDirect3DViewport2** interface.

IDirect3DDevice2::Begin

The **IDirect3DDevice2::Begin** method indicates the start of a sequence of rendered primitives. This method defines the type of these primitives and the type of vertices on which they are based. The only method you can legally call between calls to **IDirect3DDevice2::Begin** and **IDirect3DDevice2::End** is **IDirect3DDevice2::Vertex**.

```
HRESULT Begin(
  D3DPRIMITIVETYPE d3dpt,
  D3DVERTEXTYPE d3dvt,
  DWORD dwFlags
);
```

Parameters

d3dpt
One of the members of the **D3DPRIMITIVETYPE** enumerated type.

d3dvt
Indicates the type of vertices to be used in rendering this primitive. Only vertices of this type will be accepted before the corresponding **IDirect3DDevice2::End**.

This must be one of the members of the **D3DVERTEXTYPE** enumerated type, as specified in a call to the **IDirect3DDevice2::Vertex** method.

dwFlags
One or more of the following flags defining how the primitive is drawn:

D3DDP_DONOTCLIP	The application has already done the required clipping, so the system should not necessarily clip the primitives. (This flag is a hint; the system may clip the primitive even when this flag is specified, under some circumstances.)
D3DDP_DONOTUPDATEEXTENTS	Disables the updating of the screen rectangle affected by this rendering call. Using this flag can potentially help performance, but the extents returned by **IDirect3DDevice2::GetClipStatus** will not have been updated to account for the data rendered by this call.

D3DDP_OUTOFORDER	A hint to the system that the primitives can be rendered out of order. Note that back-to-back calls to DrawPrimitive methods using this flag may cause triangles from the primitives to be interleaved. The DrawPrimitive methods that use this flag are **Begin**, **BeginIndexed**, **DrawIndexedPrimitive**, and **DrawPrimitive**.
D3DDP_WAIT	Causes the method to wait until the polygons have been rendered before it returns, instead of returning as soon as the polygons have been sent to the card. (On scene-capture cards, the method returns as soon as the card responds.) This flag is typically used for debugging. Applications should not attempt to use this flag to ensure that a scene is up-to-date before continuing.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method fails if it is called after a call to the **IDirect3DDevice2::Begin** or **IDirect3DDevice2::BeginIndexed** method that has no bracketing call to **IDirect3DDevice2::End** method. Rendering calls that specify the wrong vertex type or that perform state changes will cause rendering of this primitive to fail.

This method was first introduced in the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::BeginIndexed, **IDirect3DDevice2::End**, **IDirect3DDevice2::Vertex**

IDirect3DDevice2::BeginIndexed

The **IDirect3DDevice2::BeginIndexed** method defines the start of a primitive based on indexing into an array of vertices. This method fails if it is called after a call to the **IDirect3DDevice2::Begin** or **IDirect3DDevice2::BeginIndexed** method that has no corresponding call to **IDirect3DDevice2::End**. The only method you can legally call between calls to **IDirect3DDevice2::BeginIndexed** and **IDirect3DDevice2::End** is **IDirect3DDevice2::Index**.

```
HRESULT BeginIndexed(
  D3DPRIMITIVETYPE dptPrimitiveType,
  D3DVERTEXTYPE dvtVertexType,
  LPVOID lpvVertices,
  DWORD dwNumVertices,
  DWORD dwFlags
);
```

Parameters

dptPrimitiveType
Type of primitive to be rendered by this command. This must be one of the members of the **D3DPRIMITIVETYPE** enumerated type. Note that the **D3DPT_POINTLIST** member of **D3DPRIMITIVETYPE** is not indexed.

dvtVertexType
Indicates the types of the vertices used. This must be one of the members of the **D3DVERTEXTYPE** enumerated type.

lpvVertices
Pointer to the list of vertices to be used in the primitive sequence.

dwNumVertices
Number of vertices in the above array.

dwFlags
One or more of the following flags defining how the primitive is drawn:

D3DDP_DONOTCLIP	The application has already done the required clipping, so the system should not necessarily clip the primitives. (This flag is a hint; the system may clip the primitive even when this flag is specified, under some circumstances.)
D3DDP_DONOTUPDATEEXTENTS	Disables the updating of the screen rectangle affected by this rendering call. Using this flag can potentially help performance, but the extents returned by **IDirect3DDevice2::GetClipStatus** will not have been updated to account for the data rendered by this call.
D3DDP_OUTOFORDER	A hint to the system that the primitives can be rendered out of order. Note that back-to-back calls to DrawPrimitive methods using this flag may cause triangles from the primitives to be interleaved. The DrawPrimitive methods that use this flag are **Begin**, **BeginIndexed**, **DrawIndexedPrimitive**, and **DrawPrimitive**.
D3DDP_WAIT	Causes the method to wait until the polygons have been rendered before it returns, instead of returning as soon as the polygons have been sent to the card. (On scene-capture cards, the method

returns as soon as the card responds.)

This flag is typically used for debugging. Applications should not attempt to use this flag to ensure that a scene is up-to-date before continuing.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

This method was first introduced in the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::Begin, **IDirect3DDevice2::End**, **IDirect3DDevice2::Index**

IDirect3DDevice2::BeginScene

The **IDirect3DDevice2::BeginScene** method begins a scene.

Applications must call the **IDirect3DDevice2::BeginScene** method before performing any rendering, and must call **IDirect3DDevice2::EndScene** when rendering is complete.

This method is unchanged from its implementation in the **IDirect3DDevice** interface.

```
HRESULT BeginScene();
```

Parameters

None.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error.

See Also

IDirect3DDevice2::EndScene

IDirect3DDevice2::DeleteViewport

The **IDirect3DDevice2::DeleteViewport** method removes the specified viewport from the list of viewport objects associated with the device.

```
HRESULT DeleteViewport(
  LPDIRECT3DVIEWPORT2 lpDirect3DViewport2
);
```

Parameters

lpDirect3DViewport2
: Address of the Direct3DViewport2 object that should be disassociated with this Direct3DDevice2 object.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

In the **IDirect3DDevice** interface, this method requires a pointer to an **IDirect3DViewport** interface, not an **IDirect3DViewport2** interface.

IDirect3DDevice2::DrawIndexedPrimitive

The **IDirect3DDevice2::DrawIndexedPrimitive** method renders the specified geometric primitive based on indexing into an array of vertices.

```
HRESULT DrawIndexedPrimitive(
  D3DPRIMITIVETYPE d3dptPrimitiveType,
  D3DVERTEXTYPE d3dvtVertexType,
  LPVOID lpvVertices,
  DWORD dwVertexCount,
  LPWORD dwIndices,
  DWORD dwIndexCount,
  DWORD dwFlags
);
```

Parameters

d3dptPrimitiveType

Type of primitive to be rendered by this command. This must be one of the members of the **D3DPRIMITIVETYPE** enumerated type.

Note that the **D3DPT_POINTLIST** member of **D3DPRIMITIVETYPE** is not indexed.

d3dvtVertexType
Indicates the types of the vertices used. This must be one of the members of the **D3DVERTEXTYPE** enumerated type.

lpvVertices
Pointer to the list of vertices to be used in the primitive sequence.

dwVertexCount
Defines the number of vertices in the list.

Notice that this parameter is used differently from the *dwVertexCount* parameter in the **IDirect3DDevice2::DrawPrimitive** method. In that method, the *dwVertexCount* parameter gives the number of vertices to draw, but here it gives the total number of vertices in the array pointed to by the *lpvVertices* parameter. When you call **IDirect3DDevice2::DrawIndexedPrimitive**, you specify the number of vertices to draw in the *dwIndexCount* parameter.

dwIndices
Pointer to a list of WORDs that are to be used to index into the specified vertex list when creating the geometry to render.

dwIndexCount
Specifies the number of indices provided for creating the geometry.

dwFlags
One or more of the following flags defining how the primitive is drawn:

D3DDP_DONOTCLIP	The application has already done the required clipping, so the system should not necessarily clip the primitives. (This flag is a hint; the system may clip the primitive even when this flag is specified, under some circumstances.)
D3DDP_DONOTUPDATEEXTENTS	Disables the updating of the screen rectangle affected by this rendering call. Using this flag can potentially help performance, but the extents returned by **IDirect3DDevice2::GetClipStatus** will not have been updated to account for the data rendered by this call.
D3DDP_OUTOFORDER	A hint to the system that the primitives can be rendered out of order. Note that back-to-back calls to DrawPrimitive methods using this flag may cause triangles from the primitives to be interleaved. The DrawPrimitive methods that use this flag are **Begin**, **BeginIndexed**, **DrawIndexedPrimitive**, and **DrawPrimitive**.
D3DDP_WAIT	Causes the method to wait until the polygons have been rendered before it returns, instead of

	returning as soon as the polygons have been sent to the card. (On scene-capture cards, the method returns as soon as the card responds.) This flag is typically used for debugging. Applications should not attempt to use this flag to ensure that a scene is up-to-date before continuing.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

In current versions of DirectX, **IDirect3DDevice2::DrawIndexedPrimitive** can sometimes generate an update rectangle that is larger than it strictly needs to be. If a large number of vertices need to be processed, this can have a negative impact on the performance of your application. If you are using **D3DTLVERTEX** vertices and the system is processing more vertices than you need, you should use the D3DDP_DONOTCLIP and D3DDP_DONOTUPDATEEXTENTS flags to solve the problem.

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::DrawPrimitive

IDirect3DDevice2::DrawPrimitive

The **IDirect3DDevice2::DrawPrimitive** method renders the specified array of vertices as a sequence of geometric primitives of the specified type.

```
HRESULT DrawPrimitive(
  D3DPRIMITIVETYPE dptPrimitiveType,
  D3DVERTEXTYPE dvtVertexType,
  LPVOID lpvVertices,
  DWORD dwVertexCount,
  DWORD dwFlags
);
```

Parameters

dptPrimitiveType
Type of primitive to be rendered by this command. This must be one of the members of the **D3DPRIMITIVETYPE** enumerated type.

dvtVertexType
Indicates the types of the vertices used. This must be one of the members of the **D3DVERTEXTYPE** enumerated type.

lpvVertices
Pointer to the array of vertices to be used in the primitive sequence.

dwVertexCount
Defines the number of vertices in the array.

dwFlags
One or more of the following flags defining how the primitive is drawn:

D3DDP_DONOTCLIP	The application has already done the required clipping, so the system should not necessarily clip the primitives. (This flag is a hint; the system may clip the primitive even when this flag is specified, under some circumstances.)
D3DDP_DONOTUPDATEEXTENTS	Disables the updating of the screen rectangle affected by this rendering call. Using this flag can potentially help performance, but the extents returned by **IDirect3DDevice2::GetClipStatus** will not have been updated to account for the data rendered by this call.
D3DDP_OUTOFORDER	A hint to the system that the primitives can be rendered out of order. Note that back-to-back calls to DrawPrimitive methods using this flag may cause triangles from the primitives to be interleaved. The DrawPrimitive methods that use this flag are **Begin**, **BeginIndexed**, **DrawIndexedPrimitive**, and **DrawPrimitive**.
D3DDP_WAIT	Causes the method to wait until the polygons have been rendered before it returns, instead of returning as soon as the polygons have been sent to the card. (On scene-capture cards, the method returns as soon as the card responds.) This flag is typically used for debugging. Applications should not attempt to use this flag to ensure that a scene is up-to-date before continuing.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::DrawIndexedPrimitive

IDirect3DDevice2::End

The **IDirect3DDevice2::End** method signals the completion of a primitive sequence. This method fails if no corresponding call to the **IDirect3DDevice2::Begin** method was made.

```
HRESULT End(
  DWORD dwFlags
);
```

Parameters

dwFlags
Reserved. A flag word that should be set to 0.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

This method fails if the vertex count is incorrect for the primitive type. It fails without drawing if it is called before a sufficient number of vertices is specified. If the number of Vertex or index calls made is not evenly divisible by 3 (in the case of triangles), or 2 (in the case of lineList), the remainder will be ignored.

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::Begin

IDirect3DDevice2::EndScene

The **IDirect3DDevice2::EndScene** method ends a scene that was begun by calling the **IDirect3DDevice2::BeginScene** method.

```
HRESULT EndScene();
```

Parameters

None.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error.

Remarks

When this method succeeds, the scene will have been rendered and the device surface will hold the contents of the rendering.

This method is unchanged from its implementation in the **IDirect3DDevice** interface.

See Also

IDirect3DDevice2::BeginScene

IDirect3DDevice2::EnumTextureFormats

The **IDirect3DDevice2::EnumTextureFormats** method queries the current driver for a list of supported texture formats.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DDevice** interface.

IDirect3DDevice2::GetCaps

The **IDirect3DDevice2::GetCaps** method retrieves the capabilities of the Direct3DDevice2 object.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method does not retrieve the capabilities of the display device. To retrieve this information, use the **IDirectDraw2::GetCaps** method.

This method is unchanged from its implementation in the **IDirect3DDevice** interface.

IDirect3DDevice2::GetClipStatus

The **IDirect3DDevice2::GetClipStatus** method gets the current clip status.

```
HRESULT GetClipStatus(
  LPD3DCLIPSTATUS lpD3DClipStatus
);
```

Parameters

lpD3DClipStatus
Address of a **D3DCLIPSTATUS** structure that describes the current clip status.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetClipStatus

IDirect3DDevice2::GetCurrentViewport

The **IDirect3DDevice2::GetCurrentViewport** method retrieves the current viewport.

```
HRESULT GetCurrentViewport(
  LPDIRECT3DVIEWPORT2 *lplpd3dViewport2
);
```

Parameters

lplpd3dViewport2
    Address that contains a pointer to the current viewport when the method returns. A reference is taken to the viewport object.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDPARAMS	One of the arguments is invalid.
D3DERR_NOCURRENTVIEWPORT	No current viewport has been set by a call to the **IDirect3DDevice2::SetCurrentViewport** method.

Remarks

This method increases the reference count of the viewport interface retrieved in the *lplpd3dViewport2* parameter. The application must release this interface when it is no longer needed.

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetCurrentViewport

IDirect3DDevice2::GetDirect3D

The **IDirect3DDevice2::GetDirect3D** method retrieves the current **IDirect3D2** interface.

```
HRESULT GetDirect3D(
  LPDIRECT3D2 *lplpD3D2
);
```

Parameters

lplpD3D2
    Address that will contain the interface when the method returns.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error. For a list of possible return codes, see Direct3D Immediate-Mode Return Values.

Remarks

In the **IDirect3DDevice** interface, this method retrieves the current **IDirect3D** interface instead of an **IDirect3D2** interface.

IDirect3DDevice2::GetLightState

The **IDirect3DDevice2::GetLightState** method gets a single Direct3D Device lighting-related state value.

```
HRESULT GetLightState(
  D3DLIGHTSTATETYPE dwLightStateType,
  LPDWORD lpdwLightState
);
```

Parameters

dwLightStateType
Device state variable that is being queried. This parameter can be any of the members of the **D3DLIGHTSTATETYPE** enumerated type.

lpdwLightState
Address of a variable that will contain the Direct3D Device light state when the method returns.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetLightState

IDirect3DDevice2::GetRenderState

The **IDirect3DDevice2::GetRenderState** method gets a single Direct3D Device rendering state parameter.

```
HRESULT GetRenderState(
  D3DRENDERSTATETYPE dwRenderStateType,
  LPDWORD lpdwRenderState
);
```

Parameters

dwRenderStateType
: Device state variable that is being queried. This parameter can be any of the members of the **D3DRENDERSTATETYPE** enumerated type.

lpdwRenderState
: Address of a variable that will contain the Direct3D Device render state when the method returns.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetRenderState

IDirect3DDevice2::GetRenderTarget

The **IDirect3DDevice2::GetRenderTarget** method retrieves a pointer to the DirectDraw surface that is being used as a render target.

```
HRESULT GetRenderTarget(
  LPDIRECTDRAWSURFACE *lplpRenderTarget
);
```

Parameters

lplpRenderTarget
: Address that will contain a pointer to the DirectDraw surface object that is being used as a render target by this Direct3D device.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetRenderTarget

IDirect3DDevice2::GetStats

The **IDirect3DDevice2::GetStats** method retrieves statistics about a device.

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

Parameters

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

Return Values

If the method succeeds, the return value isD3D_OK .

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DDevice** interface.

IDirect3DDevice2::GetTransform

The **IDirect3DDevice2::GetTransform** method gets a matrix describing a transformation state.

```
HRESULT GetTransform(
  D3DTRANSFORMSTATETYPE dtstTransformStateType,
  LPD3DMATRIX lpD3DMatrix
);
```

Parameters

dtstTransformStateType
Device state variable that is being modified. This parameter can be any of the members of the **D3DTRANSFORMSTATETYPE** enumerated type.

lpD3DMatrix
Address of a **D3DMATRIX** structure describing the transformation.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::SetTransform

IDirect3DDevice2::Index

The **IDirect3DDevice2::Index** method adds a new index to the currently started primitive.

```
HRESULT Index(
  WORD wVertexIndex
);
```

Parameters

wVertexIndex
Index of the next vertex to be added to the currently started primitive sequence.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material

	does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

IDirect3DDevice2::MultiplyTransform

The **IDirect3DDevice2::MultiplyTransform** method modifies the current world matrix by combining it with a specified matrix. The multiplication order is *lpD3DMatrix* times *dtstTransformStateType*.

HRESULT MultiplyTransform(
  D3DTRANSFORMSTATETYPE *dtstTransformStateType***,**
  LPD3DMATRIX *lpD3DMatrix*
);

Parameters

dtstTransformStateType
: One of the members of the **D3DTRANSFORMSTATETYPE** enumerated type. Only the D3DTRANSFORMSTATE_WORLD setting is likely to be useful. The matrix referred to by this parameter is modified by this method.

lpD3DMatrix
: Address of a **D3DMATRIX** structure that modifies the current transformation.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

An application might use the **IDirect3DDevice2::MultiplyTransform** method to work with hierarchies of transformations. For example, the geometry and transformations describing an arm might be arranged in the following hierarchy:

```
shoulder_transformation
    upper_arm geometry
    elbow transformation
        lower_arm geometry
        wrist transformation
            hand geometry
```

An application might use the following series of calls to render this hierarchy. (Not all of the parameters are shown in this pseudocode.)

```
IDirect3DDevice2::SetTransform(D3DTRANSFORMSTATE_WORLD,
    shoulder_transform)
IDirect3DDevice2::DrawPrimitive(upper_arm)
IDirect3DDevice2::MultiplyTransform(D3DTRANSFORMSTATE_WORLD,
    elbow_transform)
IDirect3DDevice2::DrawPrimitive(lower_arm)
IDirect3DDevice2::MultiplyTransform(D3DTRANSFORMSTATE_WORLD,
    wrist_transform)
IDirect3DDevice2::DrawPrimitive(hand)
```

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::DrawPrimitive, **IDirect3DDevice2::SetTransform**

IDirect3DDevice2::NextViewport

The **IDirect3DDevice2::NextViewport** method enumerates the viewports associated with the device.

```
HRESULT NextViewport(
  LPDIRECT3DVIEWPORT2 lpDirect3DViewport2,
  LPDIRECT3DVIEWPORT2 *lplpDirect3DViewport2,
  DWORD dwFlags
);
```

Parameters

lpDirect3DViewport2
: Address of a viewport in the list of viewports associated with this Direct3DDevice2 object.

lplpDirect3DViewport2
: Address of the next viewport in the list of viewports associated with this Direct3DDevice2 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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

If you attempt to retrieve the next viewport in the list when you are at the end of the list, this method returns D3D_OK but *lplpDirect3DViewport2* is NULL.

In the **IDirect3DDevice** interface, this method requires pointers to **IDirect3DViewport** interfaces, not **IDirect3DViewport2** interfaces.

IDirect3DDevice2::SetClipStatus

The **IDirect3DDevice2::SetClipStatus** method sets the current clip status.

```
HRESULT SetClipStatus(
  LPD3DCLIPSTATUS lpD3DClipStatus
);
```

Parameters

lpD3DClipStatus
: Address of a **D3DCLIPSTATUS** structure that describes the new settings for the clip status.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetClipStatus

IDirect3DDevice2::SetCurrentViewport

The **IDirect3DDevice2::SetCurrentViewport** method sets the current viewport.

```
HRESULT SetCurrentViewport(
  LPDIRECT3DVIEWPORT2 lpd3dViewport2
);
```

Parameters

lpd3dViewport2
: Address of the viewport that will become the current viewport if the method is successful.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

Applications must call this method before calling any rendering functions. Before calling this method, applications must have already called the **IDirect3DDevice2::AddViewport** method to add the viewport to the device.

Before the first call to **IDirect3DDevice2::SetCurrentViewport**, the current viewport for the device is invalid, and any attempts to render using the device will fail.

This method increases the reference count of the viewport interface specified by the *lpd3dViewport2* parameter and releases the previous viewport, if any.

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetCurrentViewport

IDirect3DDevice2::SetLightState

The **IDirect3DDevice2::SetLightState** method sets a single Direct3D Device lighting-related state value.

```
HRESULT SetLightState(
  D3DLIGHTSTATETYPE dwLightStateType,
  DWORD dwLightState
);
```

Parameters

dwLightStateType
: Device state variable that is being modified. This parameter can be any of the members of the **D3DLIGHTSTATETYPE** enumerated type.

dwLightState
: New value for the Direct3D Device light state. The meaning of this parameter is dependent on the value specified for *dwLightStateType*. For example, if *dwLightStateType* were **D3DLIGHTSTATE_COLORMODEL**, the second parameter would be one of the members of the **D3DCOLORMODEL** enumerated type.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetLightState, **IDirect3DDevice2::SetRenderState**, **IDirect3DDevice2::SetTransform**

IDirect3DDevice2::SetRenderState

The **IDirect3DDevice2::SetRenderState** method sets a single Direct3D Device rendering state parameter.

```
HRESULT SetRenderState(
  D3DRENDERSTATETYPE dwRenderStateType,
  DWORD dwRenderState
);
```

Parameters

dwRenderStateType
: Device state variable that is being modified. This parameter can be any of the members of the **D3DRENDERSTATETYPE** enumerated type.

dwRenderState
: New value for the Direct3D Device render state. The meaning of this parameter is dependent on the value specified for *dwRenderStateType*. For example, if *dwRenderStateType* were **D3DRENDERSTATE_SHADEMODE**, the second

parameter would be one of the members of the **D3DSHADEMODE** enumerated type.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetRenderState, **IDirect3DDevice2::SetLightState**, **IDirect3DDevice2::SetTransform**

IDirect3DDevice2::SetRenderTarget

The **IDirect3DDevice2::SetRenderTarget** method permits the application to easily route rendering output to a new DirectDraw surface as a render target.

```
HRESULT SetRenderTarget(
  LPDIRECTDRAWSURFACE lpNewRenderTarget,
  DWORD dwFlags
);
```

Parameters

lpNewRenderTarget
Pointer to the previously created DirectDraw surface object to which future rendering on this Direct3D Device will be directed.

dwFlags
A flag word that should be set to 0.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The error may be one of the following values:

DDERR_INVALIDPARAMS	One of the arguments is invalid.
DDERR_INVALIDSURFACETYPE	The surface passed as the first parameter is invalid.

Remarks

When you change the rendering target, all of the handles associated with the previous rendering target become invalid. This means that you will have to reacquire all of the texture handles. If you are using ramp mode, you should also update the texture handles inside materials, by calling the **IDirect3DMaterial2::SetMaterial** method. Any execute buffers (which have embedded handles) also need to be updated. The **IDirect3DDevice2::SetRenderState** method is most useful to applications that use the DrawPrimitive methods, especially when these applications do not use ramp mode.

If the new render target surface has different dimensions from the old (length, width, pixel-format), this method marks the viewport as invalid. The viewport may be revalidated after calling **IDirect3DDevice2::SetRenderTarget** by calling **IDirect3DViewport2::SetViewport** to restate viewport parameters that are compatible with the new surface.

Capabilities do not change with changes in the properties of the render target surface. Both the Direct3D HAL and the software rasterizers have only one opportunity to expose capabilities to the application. The system cannot expose different sets of capabilities depending on the format of the destination surface.

If a z-buffer is attached to the new render target, it replaces the previous z-buffer for the context. Otherwise, the old z-buffer is detached and z-buffering is disabled.

If more than one z-buffer is attached to the render target, this function fails.

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetRenderTarget

IDirect3DDevice2::SetTransform

The **IDirect3DDevice2::SetTransform** method sets a single Direct3D Device transformation-related state.

```
HRESULT SetTransform(
  D3DTRANSFORMSTATETYPE dtstTransformStateType,
  LPD3DMATRIX lpD3DMatrix
);
```

Parameters

dtstTransformStateType
: Device state variable that is being modified. This parameter can be any of the members of the **D3DTRANSFORMSTATETYPE** enumerated type.

lpD3DMatrix
Address of a **D3DMATRIX** structure that modifies the current transformation.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is an error. The method returns DDERR_INVALIDPARAMS if one of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

See Also

IDirect3DDevice2::GetTransform, **IDirect3DDevice2::SetLightState**, **IDirect3DDevice2::SetRenderState**

IDirect3DDevice2::SwapTextureHandles

The **IDirect3DDevice2::SwapTextureHandles** method swaps two texture handles.

```
HRESULT SwapTextureHandles(
  LPDIRECT3DTEXTURE2 lpD3DTex1,
  LPDIRECT3DTEXTURE2 lpD3DTex2
);
```

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error.

Remarks

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

In the **IDirect3DDevice** interface, this method requires pointers to **IDirect3DTexture** interfaces, not **IDirect3DTexture2** interfaces.

IDirect3DDevice2::Vertex

The **IDirect3DDevice2::Vertex** method adds a new Direct3D vertex to the currently started primitive.

```
HRESULT Vertex(
  LPVOID lpVertexType
);
```

Parameters

lpVertexType
Pointer to the next Direct3D vertex to be added to the currently started primitive sequence. This can be any of the Direct3D vertex types: **D3DLVERTEX**, **D3DTLVERTEX**, or **D3DVERTEX**

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following values:

D3DERR_INVALIDRAMPTEXTURE	Ramp mode is being used and the texture handle in the current material does not match the current texture handle that is set as a render state.
DDERR_INVALIDPARAMS	One of the arguments is invalid.

Remarks

This method was introduced with the **IDirect3DDevice2** interface.

IDirect3DExecuteBuffer

Applications use the methods of the **IDirect3DExecuteBuffer** interface to set up and control Direct3D execute buffers. This section is a reference to the methods of this interface. For a conceptual overview, see Execute Buffers.

The methods of the **IDirect3DExecuteBuffer** interface can be organized into the following groups:

Execute data	**GetExecuteData**
	SetExecuteData
Lock and unlock	**Lock**
	Unlock

Miscellaneous	**Initialize**
	Optimize
	Validate

The **IDirect3DExecuteBuffer** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DEXECUTEBUFFER** type is defined as a pointer to the **IDirect3DExecuteBuffer** interface:

```
typedef struct IDirect3DExecuteBuffer    *LPDIRECT3DEXECUTEBUFFER;
```

IDirect3DExecuteBuffer::GetExecuteData

The **IDirect3DExecuteBuffer::GetExecuteData** method retrieves the execute data state of the Direct3DExecuteBuffer object. The execute data is used to describe the contents of the Direct3DExecuteBuffer object.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

D3DERR_EXECUTE_LOCKED
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This call fails if the Direct3DExecuteBuffer object is locked.

See Also

IDirect3DExecuteBuffer::SetExecuteData

IDirect3DExecuteBuffer::Initialize

The **IDirect3DExecuteBuffer::Initialize** method is provided for compliance with the COM protocol.

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

Parameters

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.

Return Values

The method returns DDERR_ALREADYINITIALIZED because the Direct3DExecuteBuffer object is initialized when it is created.

IDirect3DExecuteBuffer::Lock

The **IDirect3DExecuteBuffer::Lock** method obtains a direct pointer to the commands in the execute buffer.

HRESULT Lock(
  LPD3DEXECUTEBUFFERDESC *lpDesc*
);

Parameters

lpDesc
: Address of a **D3DEXECUTEBUFFERDESC** structure. When the method returns, the **lpData** member will be set to point to the actual data to which the application has access. 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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

D3DERR_EXECUTE_LOCKED
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_WASSTILLDRAWING

Remarks

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

The **IDirect3DExecuteBuffer::Optimize** method is not currently supported.

```
HRESULT Optimize();
```

IDirect3DExecuteBuffer::SetExecuteData

The **IDirect3DExecuteBuffer::SetExecuteData** method sets the execute data state of the Direct3DExecuteBuffer object. The execute data is used to describe the contents of the Direct3DExecuteBuffer object.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK .

If the method fails, the return value may be one of the following values:

D3DERR_EXECUTE_LOCKED
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This call fails if the Direct3DExecuteBuffer object is locked.

See Also

IDirect3DExecuteBuffer::GetExecuteData

IDirect3DExecuteBuffer::Unlock

The **IDirect3DExecuteBuffer::Unlock** method 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.

```
HRESULT Unlock();
```

Parameters

None.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

D3DERR_EXECUTE_NOT_LOCKED
DDERR_INVALIDOBJECT

See Also

IDirect3DExecuteBuffer::Lock

IDirect3DExecuteBuffer::Validate

The **IDirect3DExecuteBuffer::Validate** method is not currently implemented.

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

IDirect3DLight

Applications use the methods of the **IDirect3DLight** interface to retrieve and set the capabilities of lights. This section is a reference to the methods of this interface. For a conceptual overview, see Lights.

The **IDirect3DLight** interface is obtained by calling the **IDirect3D2::CreateLight** method.

The methods of the **IDirect3DLight** interface can be organized into the following groups:

Get and set	**GetLight**
	SetLight
Initialization	**Initialize**

The **IDirect3DLight** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DLIGHT** type is defined as a pointer to the **IDirect3DLight** interface:

```
typedef struct IDirect3DLight    *LPDIRECT3DLIGHT;
```

IDirect3DLight::GetLight

The **IDirect3DLight::GetLight** method retrieves the light information for the Direct3DLight object.

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

Parameters

lpLight
Address of a **D3DLIGHT2** structure that will be filled with the current light data.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirect3DLight::SetLight

IDirect3DLight::Initialize

The **IDirect3DLight::Initialize** method is provided for compliance with the COM protocol.

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

Parameters

lpDirect3D
Address of the Direct3D structure representing the Direct3D object.

Return Values

The method returns DDERR_ALREADYINITIALIZED because the Direct3DLight object is initialized when it is created.

IDirect3DLight::SetLight

The **IDirect3DLight::SetLight** method sets the light information for the Direct3DLight object.

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

Parameters

lpLight
Address of a **D3DLIGHT2** structure that will be used to set the current light data.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirect3DLight::GetLight

IDirect3DMaterial2

Applications use the methods of the **IDirect3DMaterial2** interface to retrieve and set the properties of materials. This section is a reference to the methods of this interface. For a conceptual overview, see Materials.

The **IDirect3DMaterial2** interface is an extension of the **IDirect3DMaterial** interface. You create this interface by calling the **IDirect3D2::CreateMaterial** method.

The methods of the **IDirect3DMaterial2** interface can be organized into the following groups:

Handles	**GetHandle**
Materials	**GetMaterial**
	SetMaterial

The **IDirect3DMaterial2** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DMATERIAL2** and **LPDIRECT3DMATERIAL** types are defined as pointers to the **IDirect3DMaterial2** and **IDirect3DMaterial** interfaces:

```
typedef struct IDirect3DMaterial2    *LPDIRECT3DMATERIAL2;
typedef struct IDirect3DMaterial     *LPDIRECT3DMATERIAL;
```

IDirect3DMaterial2::GetHandle

The **IDirect3DMaterial2::GetHandle** method obtains the material handle of the Direct3DMaterial object. This handle is used in all Direct3D methods in which 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.

HRESULT GetHandle(
  LPDIRECT3DDEVICE2 *lpDirect3DDevice2*,

```
  LPD3DMATERIALHANDLE lpHandle
);
```

Parameters

lpDirect3DDevice2
: Address of the Direct3DDevice2 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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is DDERR_INVALIDOBJECT.

Remarks

In the **IDirect3DMaterial** interface, this method uses a pointer to a Direct3DMaterial object instead of a Direct3DMaterial2 object.

IDirect3DMaterial2::GetMaterial

The **IDirect3DMaterial2::GetMaterial** method retrieves the material data for the Direct3DMaterial object.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DMaterial** interface.

See Also

IDirect3DMaterial2::SetMaterial

IDirect3DMaterial::Initialize

The **IDirect3DMaterial2::Initialize** method is not implemented.

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

IDirect3DMaterial::Reserve

The **IDirect3DMaterial2::Reserve** method is not implemented.

```
HRESULT Reserve();
```

IDirect3DMaterial2::SetMaterial

The **IDirect3DMaterial2::SetMaterial** method sets the material data for the Direct3DMaterial object.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DMaterial** interface.

See Also

IDirect3DMaterial2::GetMaterial

IDirect3DMaterial::Unreserve

The **IDirect3DMaterial2::Unreserve** method is not implemented.

```
HRESULT Unreserve();
```

IDirect3DTexture2

Applications use the methods of the **IDirect3DTexture2** interface to retrieve and set the properties of textures. This section is a reference to the methods of this interface. For a conceptual overview, see Textures.

The **IDirect3DTexture2** interface is an extension of the **IDirect3DTexture** interface. You create this interface by calling the **IDirectDrawSurface::QueryInterface** method from the DirectDrawSurface object that was created as a texture map.

The methods of the **IDirect3DTexture2** interface can be organized into the following groups:

Handles	**GetHandle**
Loading	**Load**
Palette information	**PaletteChanged**

The **IDirect3DTexture2** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DTEXTURE2** and **LPDIRECT3DTEXTURE** types are defined as pointers to the **IDirect3DTexture2** and **IDirect3DTexture** interfaces:

```
typedef struct IDirect3DTexture2    *LPDIRECT3DTEXTURE2;
typedef struct IDirect3DTexture     *LPDIRECT3DTEXTURE;
```

IDirect3DTexture2::GetHandle

The **IDirect3DTexture2::GetHandle** method obtains the texture handle for the Direct3DTexture2 object. This handle is used in all Direct3D methods in which a texture is to be referenced.

```
HRESULT GetHandle(
  LPDIRECT3DDEVICE2 lpDirect3DDevice2,
  LPD3DTEXTUREHANDLE lpHandle
);
```

Parameters

lpDirect3DDevice2
: Address of the Direct3DDevice2 object into which the texture is to be loaded.

lpHandle
: Address that will contain the texture handle corresponding to the Direct3DTexture2 object.

Return Values

If the method succeeds, the return value is D3D_OK .

If the method fails, the return value may be one of the following values:

DDERR_INVALIDPARAMS

Remarks

In the **IDirect3DTexture** interface, this method uses a pointer to a Direct3DDevice object instead of a Direct3DDevice2 object.

IDirect3DTexture::Initialize

The **IDirect3DTexture2::Initialize** method is not implemented.

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

IDirect3DTexture2::Load

The **IDirect3DTexture2::Load** method loads a texture that was created with the DDSCAPS_ALLOCONLOAD flag, which indicates that memory for the DirectDraw surface is not allocated until this method loads the surface.

```
HRESULT Load(
  LPDIRECT3DTEXTURE2 lpD3DTexture2
);
```

Parameters

lpD3DTexture2
: Address of the texture to load.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error. For a list of possible return values, see Direct3D Immediate-Mode Return Values.

Remarks

In the **IDirect3DTexture** interface, this method uses a pointer to a Direct3DTexture object instead of a Direct3DTexture2 object.

See Also

IDirect3DTexture::Unload

IDirect3DTexture2::PaletteChanged

The **IDirect3DTexture2::PaletteChanged** method informs the driver that the palette has changed on a surface.

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

Parameters

dwStart
: Index of first palette entry that has changed.

dwCount
: Number of palette entries that have changed.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value is an error. For a list of possible return values, see Direct3D Immediate-Mode Return Values.

Remarks

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

This method is unchanged from its implementation in the **IDirect3DTexture** interface.

IDirect3DTexture::Unload

The **IDirect3DTexture2::Unload** method is not implemented.

```
HRESULT Unload();
```

IDirect3DViewport2

Applications use the methods of the **IDirect3DViewport2** interface to retrieve and set the properties of viewports. This section is a reference to the methods of this interface. For a conceptual overview, see Viewports and Transformations.

The **IDirect3DViewport2** interface is an extension of the **IDirect3DViewport** interface. You create the **IDirect3DViewport2** interface by calling the **IDirect3D2::CreateViewport** method.

The methods of the **IDirect3DViewport2** interface can be organized into the following groups:

Backgrounds	**GetBackground**
	GetBackgroundDepth
	SetBackground
	SetBackgroundDepth
Lights	**AddLight**
	DeleteLight
	LightElements
	NextLight
Materials and viewports	**Clear**
	GetViewport
	GetViewport2
	SetViewport
	SetViewport2

Transformation	**TransformVertices**

IDirect3DViewport2 is identical to **IDirect3DViewport** except for two new methods: **GetViewport2** and **SetViewport2**. The **IDirect3DViewport2** interface differs from the **IDirect3DViewport** interface primarily in its use of the **D3DVIEWPORT2** structure. This structure introduces a closer correspondence between window size and viewport size than is true for the **D3DVIEWPORT** structure.

The **IDirect3DViewport** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

The **LPDIRECT3DVIEWPORT2** and **LPDIRECT3DVIEWPORT** types are defined as pointers to the **IDirect3DViewport2** and **IDirect3DViewport** interfaces:

```
typedef struct IDirect3DViewport2    *LPDIRECT3DVIEWPORT2;
typedef struct IDirect3DViewport     *LPDIRECT3DVIEWPORT;
```

IDirect3DViewport2::AddLight

The **IDirect3DViewport2::AddLight** method adds the specified light to the list of Direct3DLight objects associated with this viewport.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

IDirect3DViewport2::Clear

The **IDirect3DViewport2::Clear** method clears the viewport or a set of rectangles in the viewport to the current background material.

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

Parameters

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).

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

IDirect3DViewport2::DeleteLight

The **IDirect3DViewport2::DeleteLight** method removes the specified light from the list of Direct3DLight objects associated with this viewport.

HRESULT DeleteLight(
  LPDIRECT3DLIGHT *lpDirect3DLight*
);

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

IDirect3DViewport2::GetBackground

The **IDirect3DViewport2::GetBackground** method retrieves the handle to a material that represents the current background associated with the viewport.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::SetBackground

IDirect3DViewport2::GetBackgroundDepth

The **IDirect3DViewport2::GetBackgroundDepth** method retrieves a DirectDraw surface that represents the current background-depth field associated with the viewport.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::SetBackgroundDepth

IDirect3DViewport2::GetViewport

The **IDirect3DViewport2::GetViewport** method retrieves the viewport registers of the viewport. In the **IDirect3DViewport2** interface, this method has been superseded by the **IDirect3DViewport2::GetViewport2** method.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT_dx5_DDERR_INVALIDOBJECT_ddraw
DDERR_INVALIDPARAMS_dx5_DDERR_INVALIDPARAMS_ddraw

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::GetViewport2, **IDirect3DViewport2::SetViewport**

IDirect3DViewport2::GetViewport2

The **IDirect3DViewport2::GetViewport2** method retrieves the viewport registers of the viewport.

```
HRESULT GetViewport2(
  LPD3DVIEWPORT2 lpData
);
```

Parameters

lpData
Address of a **D3DVIEWPORT2** structure representing the viewport.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirect3DViewport2::SetViewport2

IDirect3DViewport2::Initialize

The **IDirect3DViewport2::Initialize** method is not implemented.

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

IDirect3DViewport2::LightElements

The **IDirect3DViewport2::LightElements** method is not currently implemented.

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

IDirect3DViewport2::NextLight

The **IDirect3DViewport2::NextLight** method enumerates the Direct3DLight objects associated with the viewport.

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

);

Parameters

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

lplpDirect3DLight
Address of a pointer that will contain the requested light in the list of lights associated with this viewport 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.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

IDirect3DViewport2::SetBackground

The **IDirect3DViewport2::SetBackground** method sets the background associated with the viewport.

HRESULT SetBackground(
  D3DMATERIALHANDLE *hMat*
);

Parameters

hMat
Material handle that will be used as the background.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::GetBackground

IDirect3DViewport2::SetBackgroundDepth

The **IDirect3DViewport2::SetBackgroundDepth** method sets the background-depth field for the viewport.

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

Parameters

lpDDSurface
Address of the DirectDrawSurface object representing the background depth.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

The z-buffer is filled with the specified depth field when the **IDirect3DViewport2::Clear** method is called and the D3DCLEAR_ZBUFFER flag is specified. The bit depth must be 16 bits.

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::GetBackgroundDepth

IDirect3DViewport2::SetViewport

The **IDirect3DViewport2::SetViewport** method sets the viewport registers of the viewport. In the **IDirect3DViewport2** interface, this method has been superseded by the **IDirect3DViewport2::SetViewport2** method.

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

Parameters

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

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

See Also

IDirect3DViewport2::GetViewport, **IDirect3DViewport2::SetViewport2**

IDirect3DViewport2::SetViewport2

The **IDirect3DViewport2::SetViewport2** method sets the viewport registers of the viewport.

```
HRESULT SetViewport2(
  LPD3DVIEWPORT2 lpData
);
```

Parameters

lpData
: Address of a **D3DVIEWPORT2** structure that contains the new viewport.

Return Values

If the method succeeds, the return value is D3D_OK.

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirect3DViewport2::GetViewport2

IDirect3DViewport2::TransformVertices

The **IDirect3DViewport2::TransformVertices** method transforms a set of vertices by the transformation matrix.

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

Parameters

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.

Return Values

If the method succeeds, the return value is D3D_OK .

If the method fails, the return value may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

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 **D3DLVERTEX** 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 2-D 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.

This method is unchanged from its implementation in the **IDirect3DViewport** interface.

D3D_OVERLOADS

C++ programmers who define D3D_OVERLOADS can use the extensions documented here to simplify their code in Direct3D Immediate Mode applications. D3D_OVERLOADS was introduced with DirectX® 5. This section is a reference to the D3D_OVERLOADS extensions.

These extensions must be defined with C++ linkage. If D3D_OVERLOADS is defined and the inclusion of D3dtypes.h or D3d.h is surrounded by extern "C", link errors will result. For example, the following syntax would generate link errors because of C linkage of D3D_OVERLOADS functionality:

```
#define D3D_OVERLOADS
extern "C" {
#include <d3d.h>
};
```

The D3D_OVERLOADS extensions can be organized into the following groups:

Constructors	D3DLVERTEX
	D3DTLVERTEX
	D3DVECTOR
	D3DVERTEX
Operators	Access Grant Operators
	Addition Operator
	Assignment Operators
	Bitwise Equality Operator
	D3DMATRIX
	Division Operator
	Multiplication Operator
	Subtraction Operator
	Unary Operators
	Vector Dominance Operators
Helper functions	**CrossProduct**
	DotProduct
	Magnitude
	Max
	Maximize
	Min
	Minimize
	Normalize
	SquareMagnitude

D3D_OVERLOADS Constructors

This section contains reference information for the constructors provided by the D3D_OVERLOADS C++ extensions.

- **D3DLVERTEX**
- **D3DTLVERTEX**
- **D3DVECTOR**

- **D3DVERTEX**

D3DLVERTEX Constructors

The D3D_OVERLOADS constructors for the **D3DLVERTEX** structure offer a convenient way for C++ programmers to create lit vertices.

```
_D3DLVERTEX() { }
_D3DLVERTEX(const D3DVECTOR& v,
            D3DCOLOR _color, D3DCOLOR _specular,
            float _tu, float _tv)
    { x = v.x; y = v.y; z = v.z; dwReserved = 0;
      color = _color; specular = _specular;
      tu = _tu; tv = _tv;
    }
```

D3DTLVERTEX Constructors

The D3D_OVERLOADS constructors for the **D3DTLVERTEX** structure offer a convenient way for C++ programmers to create transformed and lit vertices.

```
_D3DTLVERTEX() { }
_D3DTLVERTEX(const D3DVECTOR& v, float _rhw,
             D3DCOLOR _color, D3DCOLOR _specular,
             float _tu, float _tv)
    { sx = v.x; sy = v.y; sz = v.z; rhw = _rhw;
      color = _color; specular = _specular;
      tu = _tu; tv = _tv;
    }
```

D3DVECTOR Constructors

The D3D_OVERLOADS constructors for the **D3DVECTOR** structure offer a convenient way for C++ programmers to create vectors.

```
_D3DVECTOR() { }
_D3DVECTOR(D3DVALUE f);
_D3DVECTOR(D3DVALUE _x, D3DVALUE _y, D3DVALUE _z);
_D3DVECTOR(const D3DVALUE f[3]);
```

These constructors are defined as follows:

```
inline _D3DVECTOR::_D3DVECTOR(D3DVALUE f)
    {    x = y = z = f; }

inline _D3DVECTOR::_D3DVECTOR(D3DVALUE _x, D3DVALUE _y, D3DVALUE _z)
    {    x = _x; y = _y; z = _z; }
```

```
inline _D3DVECTOR::_D3DVECTOR(const D3DVALUE f[3])
  {    x = f[0]; y = f[1]; z = f[2]; }
```

D3DVERTEX Constructors

The D3D_OVERLOADS constructors for the **D3DVERTEX** structure offer a convenient way for C++ programmers to create lit vertices.

```
_D3DVERTEX() { }
_D3DVERTEX(const D3DVECTOR& v, const D3DVECTOR& n, float _tu, float _tv)
  { x = v.x; y = v.y; z = v.z;
    nx = n.x; ny = n.y; nz = n.z;
    tu = _tu; tv = _tv;
  }
```

D3D_OVERLOADS Operators

This section contains reference information for the operators provided by the D3D_OVERLOADS C++ extensions.

- **Access Grant Operators**
- **Addition Operator**
- **Assignment Operators**
- **Bitwise Equality Operator**
- **D3DMATRIX**
- **Division Operator**
- **Multiplication Operator**
- **Subtraction Operator**
- **Unary Operators**
- **Vector Dominance Operators**

Access Grant Operators (D3D_OVERLOADS)

The bracket ("[]") operators are overloaded operators for the D3D_OVERLOADS extensions. You can use empty brackets ("[]") for access grants, "v[0]" to access the x component of a vector, "v[1]" to access the y component, and "v[2]" to access the z component. These operators are defined as follows:

```
const D3DVALUE&operator[](int i) const;
D3DVALUE&operator[](int i);
```

```
inline const D3DVALUE&
_D3DVECTOR::operator[](int i) const
{
    return (&x)[i];
}

inline D3DVALUE&
_D3DVECTOR::operator[](int i)
{
    return (&x)[i];
}
```

Addition Operator (D3D_OVERLOADS)

This binary operator is an overloaded operator for the D3D_OVERLOADS extensions. The addition operator is defined as follows:

```
_D3DVECTOR operator + (const _D3DVECTOR& v1, const _D3DVECTOR& v2);

inline _D3DVECTOR
operator + (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return _D3DVECTOR(v1.x+v2.x, v1.y+v2.y, v1.z+v2.z);
}
```

Assignment Operators (D3D_OVERLOADS)

The assignment operators are overloaded operators for the D3D_OVERLOADS extensions. Both scalar and vector forms of the "*=" and "/=" operators have been implemented. (In the vector form, multiplication and division are memberwise.)

```
_D3DVECTOR& operator += (const _D3DVECTOR& v);
_D3DVECTOR& operator -= (const _D3DVECTOR& v);
_D3DVECTOR& operator *= (const _D3DVECTOR& v);
_D3DVECTOR& operator /= (const _D3DVECTOR& v);
_D3DVECTOR& operator *= (D3DVALUE s);
_D3DVECTOR& operator /= (D3DVALUE s);
```

The assignment operators are defined as follows:

```
inline _D3DVECTOR&
_D3DVECTOR::operator += (const _D3DVECTOR& v)
{
   x += v.x;   y += v.y;   z += v.z;
   return *this;
}

inline _D3DVECTOR&
_D3DVECTOR::operator -= (const _D3DVECTOR& v)
{
   x -= v.x;   y -= v.y;   z -= v.z;
   return *this;
}

inline _D3DVECTOR&
_D3DVECTOR::operator *= (const _D3DVECTOR& v)
{
   x *= v.x;   y *= v.y;   z *= v.z;
   return *this;
}

inline _D3DVECTOR&
_D3DVECTOR::operator /= (const _D3DVECTOR& v)
{
   x /= v.x;   y /= v.y;   z /= v.z;
   return *this;
}

inline _D3DVECTOR&
_D3DVECTOR::operator *= (D3DVALUE s)
{
   x *= s;   y *= s;   z *= s;
   return *this;
}

inline _D3DVECTOR&
_D3DVECTOR::operator /= (D3DVALUE s)
{
   x /= s;   y /= s;   z /= s;
   return *this;
}
```

Bitwise Equality Operator (D3D_OVERLOADS)

This binary operator is an overloaded operator for the D3D_OVERLOADS extensions. The bitwise-equality operator is defined as follows:

```
int operator == (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

```
inline int
operator == (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return v1.x==v2.x && v1.y==v2.y && v1.z == v2.z;
}
```

D3DMATRIX (D3D_OVERLOADS)

The D3D_OVERLOADS implementation of the **D3DMATRIX** structure implements a parentheses ("()") operator. This operator offers convenient access to values in the matrix for C++ programmers. Instead of having to refer to the structure members by name, C++ programmers can refer to them by row and column number, and simply index these numbers as needed.

```
typedef struct _D3DMATRIX {
#if (defined __cplusplus) && (defined D3D_OVERLOADS)
    union {
        struct {
#endif

            D3DVALUE        _11, _12, _13, _14;
            D3DVALUE        _21, _22, _23, _24;
            D3DVALUE        _31, _32, _33, _34;
            D3DVALUE        _41, _42, _43, _44;

#if (defined __cplusplus) && (defined D3D_OVERLOADS)
        };
        D3DVALUE m[4][4];
    };
    _D3DMATRIX() { }

    D3DVALUE& operator()(int iRow, int iColumn) { return m[iRow][iColumn]; }
    const D3DVALUE& operator()(int iRow, int iColumn) const { return m[iRow][iColumn]; }
#endif
} D3DMATRIX, *LPD3DMATRIX;
```

See Also

D3DMATRIX

Division Operator (D3D_OVERLOADS)

This binary operator is an overloaded operator for the D3D_OVERLOADS extensions. Both scalar and vector forms of this operator have been implemented. The division operator is defined as follows:

```
_D3DVECTOR operator / (const _D3DVECTOR& v, D3DVALUE s);
_D3DVECTOR operator / (const _D3DVECTOR& v1, const _D3DVECTOR& v2);

inline _D3DVECTOR
operator / (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return _D3DVECTOR(v1.x/v2.x, v1.y/v2.y, v1.z/v2.z);
}

inline _D3DVECTOR
operator / (const _D3DVECTOR& v, D3DVALUE s)
{
   return _D3DVECTOR(v.x/s, v.y/s, v.z/s);
}
```

Multiplication Operator (D3D_OVERLOADS)

This binary operator is an overloaded operator for the D3D_OVERLOADS extensions. Both scalar and vector forms of this operator have been implemented. The multiplication operator is defined as follows:

```
_D3DVECTOR operator * (const _D3DVECTOR& v, D3DVALUE s);
_D3DVECTOR operator * (D3DVALUE s, const _D3DVECTOR& v);
_D3DVECTOR operator * (const _D3DVECTOR& v1, const _D3DVECTOR& v2);

inline _D3DVECTOR
operator * (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return _D3DVECTOR(v1.x*v2.x, v1.y*v2.y, v1.z*v2.z);
}

inline _D3DVECTOR
operator * (const _D3DVECTOR& v, D3DVALUE s)
{
   return _D3DVECTOR(s*v.x, s*v.y, s*v.z);
}

inline _D3DVECTOR
operator * (D3DVALUE s, const _D3DVECTOR& v)
{
   return _D3DVECTOR(s*v.x, s*v.y, s*v.z);
}
```

Subtraction Operator (D3D_OVERLOADS)

This binary operator is an overloaded operator for the D3D_OVERLOADS extensions. The subtraction operator is defined as follows:

```
_D3DVECTOR operator - (const _D3DVECTOR& v1, const _D3DVECTOR& v2);

inline _D3DVECTOR
operator - (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
  return _D3DVECTOR(v1.x-v2.x, v1.y-v2.y, v1.z-v2.z);
}
```

Unary Operators (D3D_OVERLOADS)

The unary operators are overloaded operators for the D3D_OVERLOADS extensions. The unary operators are defined as follows:

```
_D3DVECTOR operator + (const _D3DVECTOR& v);
_D3DVECTOR operator - (const _D3DVECTOR& v);

inline _D3DVECTOR
operator + (const _D3DVECTOR& v)
{
  return v;
}

inline _D3DVECTOR
operator - (const _D3DVECTOR& v)
{
  return _D3DVECTOR(-v.x, -v.y, -v.z);
}
```

Vector Dominance Operators (D3D_OVERLOADS)

These binary operators are overloaded operators for the D3D_OVERLOADS extensions. Vector v1 dominates vector v2 if any component of v1 is greater than the corresponding component of v2. Therefore, it is possible for neither of the two specified vectors to dominate the other.

```
int operator < (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
int operator <= (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

The vector-dominance operators are defined as follows:

```
inline int
operator < (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return v1[0] < v2[0] && v1[1] < v2[1] && v1[2] < v2[2];
}

inline int
operator <= (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return v1[0] <= v2[0] && v1[1] <= v2[1] && v1[2] <= v2[2];
}
```

D3D_OVERLOADS Helper Functions

This section contains reference information for the helper functions provided by the D3D_OVERLOADS C++ extensions.

- **CrossProduct**
- **DotProduct**
- **Magnitude**
- **Max**
- **Maximize**
- **Min**
- **Minimize**
- **Normalize**
- **SquareMagnitude**

CrossProduct

This helper function returns the cross product of the specified vectors. **CrossProduct** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
_D3DVECTOR CrossProduct (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

This function is defined as follows:

```
inline _D3DVECTOR
CrossProduct (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
    _D3DVECTOR result;
    result[0] = v1[1] * v2[2] - v1[2] * v2[1];
    result[1] = v1[2] * v2[0] - v1[0] * v2[2];
    result[2] = v1[0] * v2[1] - v1[1] * v2[0];

    return result;
}
```

See Also

DotProduct

DotProduct

This helper function returns the dot product of the specified vectors. **DotProduct** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
D3DVALUE DotProduct (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

This function is defined as follows:

```
inline D3DVALUE
DotProduct (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return v1.x*v2.x + v1.y * v2.y + v1.z*v2.z;
}
```

See Also

CrossProduct

Magnitude

This helper function returns the absolute value of the specified vector. **Magnitude** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
D3DVALUE Magnitude (const _D3DVECTOR& v);
```

This function is defined as follows:

```
inline D3DVALUE
Magnitude (const _D3DVECTOR& v)
{
   return (D3DVALUE) sqrt(SquareMagnitude(v));
}
```

See Also

SquareMagnitude

Max

This helper function returns the maximum component of the specified vector. **Max** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
D3DVALUE Max (const _D3DVECTOR& v);
```

This function is defined as follows:

```
inline D3DVALUE
Max (const _D3DVECTOR& v)
{
   D3DVALUE ret = v.x;
   if (ret < v.y) ret = v.y;
   if (ret < v.z) ret = v.z;
   return ret;
}
```

See Also

Min

Maximize

This helper function returns a vector that is made up of the largest components of the two specified vectors. **Maximize** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
_D3DVECTOR Maximize (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

This function is defined as follows:

```
inline _D3DVECTOR
Maximize (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return _D3DVECTOR( v1[0] > v2[0] ? v1[0] : v2[0],
                 v1[1] > v2[1] ? v1[1] : v2[1],
                 v1[2] > v2[2] ? v1[2] : v2[2]);
}
```

Remarks

You could use the **Maximize** and **Minimize** functions to compute the bounding box for a set of points, in a function that looks like this:

```
  void
  ComputeBoundingBox(const D3DVECTOR *pts, int N, D3DVECTOR *min, D3DVECTOR
*max)
  {
      int i;
      *min = *max = pts[0];
      for (i = 1; i < N; i += 1)
      {
          *min = Minimize(*min, pts[i]);
          *max = Maximize(*max, pts[i]);
      }
  }
```

See Also

Minimize

Min

This helper function returns the minimum component of the specified vector. **Min** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
D3DVALUE Min (const _D3DVECTOR& v);
```

This function is defined as follows:

```
inline D3DVALUE
Min (const _D3DVECTOR& v)
{
    D3DVALUE ret = v.x;
    if (v.y < ret) ret = v.y;
    if (v.z < ret) ret = v.z;
    return ret;
}
```

See Also

Max

Minimize

This helper function returns a vector that is made up of the smallest components of the two specified vectors. **Minimize** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
_D3DVECTOR Minimize (const _D3DVECTOR& v1, const _D3DVECTOR& v2);
```

This function is defined as follows:

```
inline _D3DVECTOR
Minimize (const _D3DVECTOR& v1, const _D3DVECTOR& v2)
{
   return _D3DVECTOR( v1[0] < v2[0] ? v1[0] : v2[0],
                v1[1] < v2[1] ? v1[1] : v2[1],
                v1[2] < v2[2] ? v1[2] : v2[2]);
}
```

Remarks

You could use the **Maximize** and **Minimize** functions to compute the bounding box for a set of points, in a function that looks like this:

```
    void
    ComputeBoundingBox(const D3DVECTOR *pts, int N, D3DVECTOR *min, D3DVECTOR
*max)
    {
        int i;
        *min = *max = pts[0];
        for (i = 1; i < N; i += 1)
        {
            *min = Minimize(*min, pts[i]);
            *max = Maximize(*max, pts[i]);
        }
    }
```

See Also

Maximize

Normalize

This helper function returns the normalized version of the specified vector (that is, a unit-length vector with the same direction as the source). **Normalize** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
_D3DVECTOR Normalize (const _D3DVECTOR& v);
```

This function is defined as follows:

```
inline _D3DVECTOR
Normalize (const _D3DVECTOR& v)
{
   return v / Magnitude(v);
}
```

SquareMagnitude

This helper function returns the square of the absolute value of the specified vector. **SquareMagnitude** is part of the suite of extra functionality that is available to C++ programmers who define D3D_OVERLOADS.

```
D3DVALUE SquareMagnitude (const _D3DVECTOR& v);
```

This function is defined as follows:

```
inline D3DVALUE
SquareMagnitude (const _D3DVECTOR& v)
{
   return v.x*v.x + v.y*v.y + v.z*v.z;
}
```

See Also

Magnitude

Macros

This section contains reference information for the macros provided by Direct3D's Immediate Mode.

- **D3DDivide**
- **D3DMultiply**
- **D3DRGB**
- **D3DRGBA**
- **D3DSTATE_OVERRIDE**
- **D3DVAL**
- **D3DVALP**
- **RGB_GETBLUE**
- **RGB_GETGREEN**

- **RGB_GETRED**
- **RGB_MAKE**
- **RGB_TORGBA**
- **RGBA_GETALPHA**
- **RGBA_GETBLUE**
- **RGBA_GETGREEN**
- **RGBA_GETRED**
- **RGBA_MAKE**
- **RGBA_SETALPHA**
- **RGBA_TORGB**

D3DDivide

The **D3DDivide** macro divides two values.

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

Parameters

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

Return Values

The macros returns the quotient of the division.

See Also

D3DMultiply

D3DMultiply

The **D3DMultiply** macro multiplies two values.

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

Parameters

a and *b*
    Values to be multiplied.

Return Values

The macros returns the product of the multiplication.

See Also

D3DDivide

D3DRGB

The **D3DRGB** macro initializes a color with the supplied RGB values.

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

Parameters

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

Return Values

The macros returns the **D3DCOLOR** value corresponding to the supplied RGB values.

See Also

D3DRGBA

D3DRGBA

The **D3DRGBA** macro initializes a color with the supplied RGBA values.

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

Parameters

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

Return Values

The macros returns the **D3DCOLOR** value corresponding to the supplied RGBA values.

See Also

D3DRGB

D3DSTATE_OVERRIDE

The **D3DSTATE_OVERRIDE** macro overrides the state of the rasterization, lighting, or transformation module. Applications can use this macro to lock and unlock a state.

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

Parameters

type

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

Return Values

No return value.

Remarks

An application might, for example, use the **STATE_DATA** macro (defined in the D3dmacs.h header file in the Misc directory of the DirectX 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

The **D3DVAL** macro creates a value whose type is **D3DVALUE**.

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

Parameters

val
: Value to be converted.

Return Values

The macros returns the converted value.

See Also

D3DVALP

D3DVALP

The **D3DVALP** macro creates a value of the specified precision.

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

Parameters

val
: Value to be converted.

prec
: Ignored.

Return Values

The macros returns the converted value.

Remarks

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

See Also

D3DVAL

RGB_GETBLUE

The **RGB_GETBLUE** macro retrieves the blue component of a **D3DCOLOR** value.

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

Parameters

rgb
: Color index from which the blue component is retrieved.

Return Values

Returns the blue component.

RGB_GETGREEN

The **RGB_GETGREEN** macro retrieves the green component of a **D3DCOLOR** value.

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

Parameters

rgb
: Color index from which the green component is retrieved.

Return Values

The macros returns the green component.

RGB_GETRED

The **RGB_GETRED** macro retrieves the red component of a **D3DCOLOR** value.

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

Parameters

rgb
: Color index from which the red component is retrieved.

Return Values

The macros returns the red component.

RGB_MAKE

The **RGB_MAKE** macro creates an RGB color from supplied values.

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

Parameters

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.

Return Values

The macros returns the color.

RGB_TORGBA

The **RGB_TORGBA** macro creates an RGBA color from a supplied RGB color.

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

Parameters

rgb
: RGB color to be converted to an RGBA color.

Return Values

Returns the RGBA color.

See Also

RGBA_TORGB

RGBA_GETALPHA

The **RGB_GETALPHA** macro retrieves the alpha component of an RGBA **D3DCOLOR** value.

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

Parameters

rgb

Color index from which the alpha component is retrieved.

Return Values

The macros returns the alpha component.

RGBA_GETBLUE

The **RGBA_GETBLUE** macro retrieves the blue component of an RGBA **D3DCOLOR** value.

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

Parameters

rgb
Color index from which the blue component is retrieved.

Return Values

The macros returns the blue component.

RGBA_GETGREEN

The **RGBA_GETGREEN** macro retrieves the green component of an RGBA **D3DCOLOR** value.

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

Parameters

rgb
Color index from which the green component is retrieved.

Return Values

The macros returns the green component.

RGBA_GETRED

The **RGBA_GETRED** macro retrieves the red component of an RGBA **D3DCOLOR** value.

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

Parameters

rgb
: Color index from which the red component is retrieved.

Return Values

The macros returns the red component.

RGBA_MAKE

The **RGBA_MAKE** macro creates an RGBA **D3DCOLOR** value from supplied red, green, blue, and alpha components.

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

Parameters

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

Return Values

The macros returns the color.

RGBA_SETALPHA

The **RGBA_SETALPHA** macro sets the alpha component of an RGBA **D3DCOLOR** value.

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

Parameters

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

x
: Value of alpha component to be set.

Return Values

The macros returns the RGBA color whose alpha component has been set.

RGBA_TORGB

The **RGBA_TORGB** macro creates an RGB **D3DCOLOR** value from a supplied RGBA **D3DCOLOR** value by stripping off the alpha component of the color.

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

Parameters

rgba
RGBA color to be converted to an RGB color.

Return Values

The macros returns the RGB color.

See Also

RGB_TORGBA

Callback Functions

This section contains reference information for the callback functions you may need to implement when you work with Direct3D Immediate Mode.

- **D3DENUMDEVICESCALLBACK**
- **D3DENUMTEXTUREFORMATSCALLBACK**
- **D3DVALIDATECALLBACK**

D3DENUMDEVICESCALLBACK

D3DENUMDEVICESCALLBACK is the prototype definition for the callback function to enumerate installed Direct3D devices.

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

Parameters

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 of a **D3DDEVICEDESC** structure that contains the hardware capabilities of the Direct3D device.

lpD3DHELDeviceDesc
: Address of a **D3DDEVICEDESC** structure that contains the emulated capabilities of the Direct3D device.

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

Return Values

Applications should return one of the following values:

D3DENUMRET_CANCEL
: Cancel the enumeration.

D3DENUMRET_OK
: Continue the enumeration.

Remarks

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

D3DENUMTEXTUREFORMATSCALLBACK is the prototype definition for the callback function to enumerate texture formats.

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

Parameters

lpDdsd
Address of a **DDSURFACEDESC** structure containing the texture information.

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

Return Values

Applications should return one of the following values:

D3DENUMRET_CANCEL
Cancel the enumeration.

D3DENUMRET_OK
Continue the enumeration.

Remarks

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

D3DVALIDATECALLBACK is the application-defined callback function supplied when an application calls the **IDirect3DExecuteBuffer::Validate** method. The **IDirect3DExecuteBuffer::Validate** method is not currently implemented.

IDirect3DExecuteBuffer::Validate is a debugging routine that checks the execute buffer and returns an offset into the buffer when any errors are encountered.

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

Parameters

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

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

Remarks

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.

Structures

This section contains information about the following structures used with Direct3D Immediate Mode.

- **D3DBRANCH**
- **D3DCLIPSTATUS**
- **D3DCOLORVALUE**
- **D3DDEVICEDESC**
- **D3DEXECUTEBUFFERDESC**
- **D3DEXECUTEDATA**
- **D3DFINDDEVICERESULT**
- **D3DFINDDEVICESEARCH**
- **D3DHVERTEX**
- **D3DINSTRUCTION**
- **D3DLIGHT2**
- **D3DLIGHTDATA**
- **D3DLIGHTINGCAPS**
- **D3DLIGHTINGELEMENT**
- **D3DLINE**
- **D3DLINEPATTERN**
- **D3DLVERTEX**
- **D3DMATERIAL**
- **D3DMATRIX**
- **D3DMATRIXLOAD**
- **D3DMATRIXMULTIPLY**
- **D3DPICKRECORD**
- **D3DPOINT**
- **D3DPRIMCAPS**
- **D3DPROCESSVERTICES**
- **D3DRECT**
- **D3DSPAN**
- **D3DSTATE**

- **D3DSTATS**
- **D3DSTATUS**
- **D3DTEXTURELOAD**
- **D3DTLVERTEX**
- **D3DTRANSFORMCAPS**
- **D3DTRANSFORMDATA**
- **D3DTRIANGLE**
- **D3DVECTOR**
- **D3DVERTEX**
- **D3DVIEWPORT**
- **D3DVIEWPORT2**

D3DBRANCH

The **D3DBRANCH** structure performs conditional operations inside an execute buffer. This structure is a forward-branch structure.

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

Members

dwMask
Bitmask for the branch. This mask is combined with the driver-status mask by using the bitwise **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.

D3DCLIPSTATUS

The **D3DCLIPSTATUS** structure describes the current clip status and extents of the clipping region. This structure was introduced in DirectX 5.

```
typedef struct _D3DCLIPSTATUS {
    DWORD dwFlags;
    DWORD dwStatus;
    float minx, maxx;
    float miny, maxy;
    float minz, maxz;
} D3DCLIPSTATUS, *LPD3DCLIPSTATUS;
```

Members

dwFlags
Flags describing whether this structure describes 2-D extents, 3-D extents, or the clip status. This member can be a combination of the following flags:

D3DCLIPSTATUS_STATUS
The structure describes the current clip status.

D3DCLIPSTATUS_EXTENTS2
The structure describes the current 2-D extents. This flag cannot be combined with D3DCLIPSTATUS_EXTENTS3.

D3DCLIPSTATUS_EXTENTS3
The structure describes the current 3-D extents. This flag cannot be combined with D3DCLIPSTATUS_EXTENTS2.

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

minx, maxx, miny, maxy, minz, maxz
x, y, and z extents of the current clipping region.

See Also

IDirect3DDevice2::GetClipStatus, **IDirect3DDevice2::SetClipStatus**

D3DCOLORVALUE

The **D3DCOLORVALUE** structure describes color values for the **D3DLIGHT2** and **D3DMATERIAL** structures.

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

Members

dvR, **dvG**, **dvB**, and **dvA**
Values of the **D3DVALUE** type specifying the red, green, blue, and alpha components of a color. These values generally range from 0 to 1, with 0 being black.

Remarks

You can set the members of this structure to values outside the range of 0 to 1 to implement some unusual effects. Values greater than 1 produce strong lights that tend to wash out a scene. Negative values produce dark lights, which actually remove light from a scene. For more information, see Colored Lights.

D3DDEVICEDESC

The **D3DDEVICEDESC** structure contains a description of the current device. This structure is used to query the current device by such methods as **IDirect3DDevice2::GetCaps**.

```
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;
    DWORD          dwMinTextureWidth, dwMinTextureHeight;
    DWORD          dwMaxTextureWidth, dwMaxTextureHeight;
    DWORD          dwMinStippleWidth, dwMaxStippleWidth;
    DWORD          dwMinStippleHeight, dwMaxStippleHeight;
} D3DDEVICEDESC, *LPD3DDEVICEDESC;
```

Members

dwSize

Size, in bytes, of this structure. You can use the **D3DDEVICEDESCSIZE** constant for this value. This member must be initialized before the structure is used.

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_DEVICERENDERBITDEPTH

The **dwDeviceRenderBitDepth** member is valid.

D3DDD_DEVICEZBUFFERBITDEPTH

The **dwDeviceZBufferBitDepth** 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_CANRENDERAFTERFLIP

Device can queue rendering commands after a page flip. Applications should not change their behavior if this flag is set; this capability simply means that the device is relatively fast.

This flag was introduced in DirectX 5.

D3DDEVCAPS_DRAWPRIMTLVERTEX

Device exports a DrawPrimitive-aware HAL.

This flag was introduced in DirectX 5.

D3DDEVCAPS_EXECUTESYSTEMMEMORY

Device can use execute buffers from system memory.

D3DDEVCAPS_EXECUTEVIDEOMEMORY

Device can use execute buffer 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_TEXTURENONLOCALVIDMEM

Device can retrieve textures from nonlocal video (AGP) memory.

This flag was introduced in DirectX 5. For more information about AGP memory, see Using Non-local Video Memory Surfaces in the DirectDraw documentation.

D3DDEVCAPS_TEXTURESYSTEMMEMORY

Device can retrieve textures from system memory.

D3DDEVCAPS_TEXTUREVIDEOMEMORY

Device can retrieve 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 3-D 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 or more 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.

dwMinTextureWidth, dwMinTextureHeight
Minimum texture width and height for this device. These members were introduced in DirectX 5.

dwMaxTextureWidth, dwMaxTextureHeight
Maximum texture width and height for this device. These members were introduced in DirectX 5.

dwMinStippleWidth, dwMaxStippleWidth
Minimum and maximum width of the stipple pattern for this device. These members were introduced in DirectX 5.

dwMinStippleHeight, dwMaxStippleHeight
Minimum and maximum height of the stipple pattern for this device. These members were introduced in DirectX 5.

See Also

D3DCOLORMODEL, **D3DFINDDEVICERESULT**, **D3DLIGHTINGCAPS**, **D3DPRIMCAPS**, **D3DTRANSFORMCAPS**

D3DEXECUTEBUFFERDESC

The **D3DEXECUTEBUFFERDESC** structure describes the execute buffer for such methods as **IDirect3DDevice::CreateExecuteBuffer** and **IDirect3DExecuteBuffer::Lock**.

```
typedef struct _D3DExecuteBufferDesc {
    DWORD  dwSize;
    DWORD  dwFlags;
    DWORD  dwCaps;
    DWORD  dwBufferSize;
    LPVOID lpData;
} D3DEXECUTEBUFFERDESC;
typedef D3DEXECUTEBUFFERDESC *LPD3DEXECUTEBUFFERDESC;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before the structure is used.

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

D3DDEBCAPS_SYSTEMMEMORY
The execute buffer data resides in system memory.

D3DDEBCAPS_VIDEOMEMORY
The execute buffer data resides in device memory.

dwBufferSize
Size of the execute buffer, in bytes.

lpData
Address of the buffer data.

D3DEXECUTEDATA

The **D3DEXECUTEDATA** structure 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.

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

Members

dwSize
Size of this structure, in bytes. This member must be initialized before the structure is used.

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

The **D3DFINDDEVICERESULT** structure identifies a device an application has found by calling the **IDirect3D2::FindDevice** method.

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

Members

dwSize
Size, in bytes, of the structure. This member must be initialized before the structure is used.

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

The **D3DFINDDEVICESEARCH** structure specifies the characteristics of a device an application wants to find. This structure is used in calls to the **IDirect3D2::FindDevice** method.

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

Members

dwSize
Size, in bytes, of this structure. This member must be initialized before the structure is used.

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
Reserved.

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

See Also

D3DFINDDEVICERESULT

D3DHVERTEX

The **D3DHVERTEX** structure defines a homogeneous vertex used when the application is supplying screen coordinate data that needs clipping. This structure is part of the **D3DTRANSFORMDATA** structure.

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

Members

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

The **D3DINSTRUCTION** structure 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.

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

Members

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.

D3DLIGHT2

The **D3DLIGHT2** structure defines the light type in calls to methods such as **IDirect3DLight::SetLight** and **IDirect3DLight::GetLight**.

For DirectX 5, this structure supersedes the **D3DLIGHT** structure. **D3DLIGHT2** is identical to **D3DLIGHT** except for the addition of the **dwFlags** member. In addition, the **dvAttenuation** members are interpreted differently in **D3DLIGHT2** than they were for **D3DLIGHT**.

```
typedef struct _D3DLIGHT2 {
    DWORD         dwSize;
    D3DLIGHTTYPE  dltType;
    D3DCOLORVALUE dcvColor;
    D3DVECTOR     dvPosition;
    D3DVECTOR     dvDirection;
    D3DVALUE      dvRange;
    D3DVALUE      dvFalloff;
    D3DVALUE      dvAttenuation0;
    D3DVALUE      dvAttenuation1;
    D3DVALUE      dvAttenuation2;
    D3DVALUE      dvTheta;
    D3DVALUE      dvPhi;
    DWORD         dwFlags;      // new member for DirectX 5
} D3DLIGHT2, *LPD3DLIGHT2;
```

Members

dwSize
Size, in bytes, of this structure. You must specify a value for this member. Direct3D uses the specified size to determine whether this is a **D3DLIGHT** or a **D3DLIGHT2** 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. In ramp mode, the color is converted to a gray scale.

dvPosition
Position of the light in world space. This member has no meaning for directional lights and is ignored in that case.

dvDirection
Direction the light is pointing in world space. This member only has meaning for directional and spotlights. This vector need not be normalized but it should have a non-zero length.

dvRange
Distance beyond which the light has no effect. The maximum allowable value for this member is D3DLIGHT_RANGE_MAX, which is defined as the square root of FLT_MAX. This member does not affect directional lights.

dvFalloff

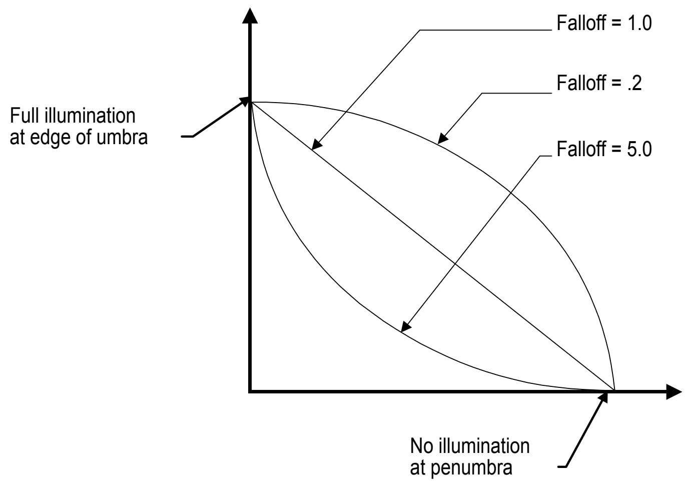
Decrease in illumination between a spotlight's umbra (the angle specified by the **dvTheta** member) and the outer edge of the penumbra (the angle specified by the **dvPhi** member). This feature was implemented for DirectX 5.

The intensity of the light at any point in the penumbra is described by the following equation:

$$Light \times \cos^{falloff}\left[\frac{\pi}{2}\left|\frac{2\,rho - dvTheta}{dvPhi - dvTheta}\right|\right]$$

In this equation, *rho* is the angle between the axis of the spotlight and the illuminated point.

A value of 1.0 specifies linear falloff from the umbra to the penumbra. If the value is anything other than 1.0, it is used as an exponent to shape the curve. Values greater than 1.0 cause the light to fall off quickly at first and then fade slowly to the penumbra. Values which are less than 1.0 create the opposite effect. The following graph shows the affect of changing these values:



The effect of falloff on the lighting is subtle. Furthermore, a small performance penalty is incurred by shaping the falloff curve. For these reasons, most developers set this value to 1.0.

dvAttenuation0 through **dvAttenuation2**

Values specifying how a light's intensity changes over distance. (Attenuation does not affect directional lights.) In the **D3DLIGHT2** structure these values are interpreted differently than they were for the **D3DLIGHT** structure.

The distance from the light to the vertex is normalized to the range by the following formula:

```
distance = (range-distance)/range
```

This results in the distance value being from 1.0 at the light to 0.0 at the light’s full range. Then the combined intensity factor of the light is calculated using the following formula:

```
intensity = dvAttenuation0 +
   dvAttenuation1 * distance +
   dvAttenuation2 * distance squared
```

This intensity factor is then multiplied by the light color to produce the final intensity of the light.

Setting the attenuation values to 1,0,0 produces a light that doesn’t change over distance. Setting the values to 0,1,0 produces a light that is at full intensity at the light, zero intensity at the light’s range, and that declines linearly between the two extremes. The values 0,0,1 produce a light that mimics the standard "1/distance squared" falloff rate that should be familiar from introductory physics classes. (The differences in this last case are that the curve is softer and that the intensity of the light doesn’t go to infinity at the light source.)

You can use various combinations of values to create unique lights. You can even use negative values—this is another way to achieve a dark light effect. Just as when you use negative values for colors, when you are in ramp mode you cannot use dark lights to produce anything darker than the current setting for the ambient light.

dvTheta
Angle, in radians, of the spotlight's umbra—that is, the fully illuminated spotlight cone. This value must be less than pi radians.

dvPhi
Angle, in radians, defining the outer edge of the spotlight's penumbra. Points outside this cone are not lit by the spotlight. This value must be between 0 and the value specified for the **dvTheta** member.

dwFlags
A combination of the following performance-related flags. This member is new for DirectX 5.

D3DLIGHT_ACTIVE	Enables the light. This flag must be set to enable the light; if it is not set, the light is ignored.
D3DLIGHT_NO_SPECULAR	Turns off specular highlights for the light.

Remarks

In the **D3DLIGHT** structure, the affects of the attenuation settings were difficult to predict; developers were encouraged to experiment with the settings until they achieved the desired result. For **D3DLIGHT2**, it is much easier to work with lighting attenuation.

For more information about lights, see Lights and IDirect3DLight.

See Also

D3DLIGHTTYPE

D3DLIGHTDATA

The **D3DLIGHTDATA** structure describes the points to be lit and resulting colors in calls to the **IDirect3DViewport2::LightElements** method.

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

Members

dwSize
: Size, in bytes, of this structure. This member must be initialized before the structure is used.

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

The **D3DLIGHTINGCAPS** structure describes the lighting capabilities of a device. This structure is a member of the **D3DDEVICEDESC** structure.

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

Members

dwSize
Size, in bytes, of this structure. This member must be initialized before the structure is used.

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

D3DLIGHTCAPS_DIRECTIONAL
Supports directional lights.

D3DLIGHTCAPS_PARALLELPOINT
Supports parallel point lights.

D3DLIGHTCAPS_POINT
Supports point lights.

D3DLIGHTCAPS_SPOT
Supports spotlights.

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

The **D3DLIGHTINGELEMENT** structure describes the points in model space that will be lit. This structure is part of the **D3DLIGHTDATA** structure.

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

Members

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, **IDirect3DViewport2::LightElements**

D3DLINE

The **D3DLINE** structure describes a line for the **D3DOP_LINE** opcode in the **D3DOPCODE** enumerated type.

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

Members

wV1 and **wV2**
    Vertex indices.

Remarks

The instruction count defines the number of line segments.

D3DLINEPATTERN

The **D3DLINEPATTERN** structure describes a line pattern. These values are used by the **D3DRENDERSTATE_LINEPATTERN** render state in the **D3DRENDERSTATETYPE** enumerated type.

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

Members

wRepeatFactor

Number of times to duplicate each series of 1s and 0s specified in the **wLinePattern** member. This repeat factor allows an application to "stretch" the line pattern.

wLinePattern

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

Remarks

A line pattern specifies how a line is drawn. The line pattern is always the same, no matter where it is started. (This is as opposed to stippling, which affects how objects are rendered; that is, to imitate transparency.)

The line pattern specifies up to a 16-pixel pattern of on and off pixels along the line. The **wRepeatFactor** member specifies how many pixels are repeated for each entry in **wLinePattern**.

D3DLVERTEX

The **D3DLVERTEX** structure defines an untransformed and lit vertex (model coordinates with color). An application should use this structure when the vertex transformations will be handled by Direct3D. This structure contains only data and a color that would be filled by software lighting.

```
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;
```

Members

dvX, **dvY**, and **dvZ**
Values of the **D3DVALUE** type specifying the model 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.

See Also

D3DTLVERTEX, **D3DVERTEX**

D3DMATERIAL

The **D3DMATERIAL** structure specifies material properties in calls to the **IDirect3DMaterial2::GetMaterial** and **IDirect3DMaterial2::SetMaterial** methods.

```
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;
```

Members

dwSize
: Size, in bytes, of this structure. This member must be initialized before the structure is used.

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.

Remarks

The texture handle specified by the **hTexture** member 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.

To turn off specular highlights for a material, you must set the **dvPower** member to 0—simply setting the specular color components to 0 is not enough.

See Also

IDirect3DMaterial2::GetMaterial, **IDirect3DMaterial2::SetMaterial**

D3DMATRIX

The **D3DMATRIX** structure describes a matrix for such methods as **IDirect3DDevice::GetMatrix** and **IDirect3DDevice::SetMatrix**.

C++ programmers can use an extended version of this structure that includes a parentheses ("()") operator. For more information, see **D3DMATRIX (D3D_OVERLOADS)**

```
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;
```

Remarks

In Direct3D, the _34 element of a projection matrix cannot be a negative number. If your application needs to use a negative value in this location, it should scale the entire projection matrix by -1, instead.

See Also

IDirect3DDevice::GetMatrix, **IDirect3DDevice::SetMatrix**

D3DMATRIXLOAD

The **D3DMATRIXLOAD** structure describes the operand data for the **D3DOP_MATRIXLOAD** opcode in the **D3DOPCODE** enumerated type.

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

Members

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

See Also

D3DOPCODE

D3DMATRIXMULTIPLY

The **D3DMATRIXMULTIPLY** structure describes the operand data for the **D3DOP_MATRIXMULTIPLY** opcode in the **D3DOPCODE** enumerated type.

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

Members

hDestMatrix
: Handle to 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

The **D3DPICKRECORD** structure returns information about picked primitives in an execute buffer for the **IDirect3DDevice::GetPickRecords** method.

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

Members

bOpcode
: Opcode of the picked primitive.

bPad
: Pad byte.

dwOffset

Offset from the start of the instruction segment portion of the execute buffer in which the picked primitive was found. (The instruction segment portion of the execute buffer is the part of the execute buffer that follows the vertex list.)

dvZ
Depth of the picked primitive.

Remarks

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

The **D3DPOINT** structure describes operand data for the **D3DOP_POINT** opcode in the in **D3DOPCODE** enumerated type.

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

Members

wCount
Number of points.

wFirst
Index of the first vertex.

Remarks

Points are rendered by using a list of vertices.

See Also

D3DOPCODE

D3DPRIMCAPS

The **D3DPRIMCAPS** structure 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.

```
typedef struct _D3DPrimCaps {
    DWORD dwSize;               // size of structure
    DWORD dwMiscCaps;           // miscellaneous caps
    DWORD dwRasterCaps;         // raster caps
    DWORD dwZCmpCaps;           // z-comparison caps
    DWORD dwSrcBlendCaps;       // source blending caps
    DWORD dwDestBlendCaps;      // destination blending caps
    DWORD dwAlphaCmpCaps;       // alpha-test comparison caps
    DWORD dwShadeCaps;          // shading caps
    DWORD dwTextureCaps;        // texture caps
    DWORD dwTextureFilterCaps;  // texture filtering caps
    DWORD dwTextureBlendCaps;   // texture blending caps
    DWORD dwTextureAddressCaps; // texture addressing caps
    DWORD dwStippleWidth;       // stipple width
    DWORD dwStippleHeight;      // stipple height
} D3DPRIMCAPS, *LPD3DPRIMCAPS;
```

Members

dwSize

Size, in bytes, of this structure. This member must be initialized before the structure is used.

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_ANISOTROPY

The device supports anisotropic filtering. For more information, see **D3DRENDERSTATE_ANISOTROPY** in the **D3DRENDERSTATETYPE** structure.

This flag was introduced in DirectX 5.

D3DPRASTERCAPS_ANTIALIASEDGES

The device can antialias lines forming the convex outline of objects. For more information, see **D3DRENDERSTATE_EDGEANTIALIAS** in the **D3DRENDERSTATETYPE** structure.

This flag was introduced in DirectX 5.

D3DPRASTERCAPS_ANTIALIASSORTDEPENDENT

The device supports antialiasing that is dependent on the sort order of the polygons (back-to-front or front-to-back). The application must draw polygons in the right order for antialiasing to occur. For more information, see the **D3DANTIALIASMODE** enumerated type.

This flag was introduced in DirectX 5.

D3DPRASTERCAPS_ANTIALIASSORTINDEPENDENT

The device supports antialiasing that is not dependent on the sort order of the polygons. For more information, see the **D3DANTIALIASMODE** enumerated type.

This flag was introduced in DirectX 5.

D3DPRASTERCAPS_DITHER

The device can dither to improve color resolution.

D3DPRASTERCAPS_FOGRANGE

The device supports range-based fog. In range-based fog, the distance of an object from the viewer is used to compute fog effects, not the depth of the object (that is, the z-coordinate) in the scene. For more information, see **D3DRENDERSTATE_RANGEFOGENABLE**.

This flag was introduced in DirectX 5.

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_MIPMAPLODBIAS

The device supports level-of-detail (LOD) bias adjustments. These bias adjustments enable an application to make a mipmap appear crisper or less sharp than it normally would. For more information about LOD bias in mipmaps, see **D3DRENDERSTATE_MIPMAPLODBIAS.**

This flag was introduced in DirectX 5.

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 scan line. 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_ZBIAS

The device supports z-bias values. These are integer values assigned to polygons that allow physically coplanar polygons to appear separate. For more information, see **D3DRENDERSTATE_ZBIAS** in the **D3DRENDERSTATETYPE** structure.

This flag was introduced in DirectX 5.

D3DPRASTERCAPS_ZBUFFERLESSHSR

The device can perform hidden-surface removal without requiring the application to sort polygons, and without requiring the allocation of a z-buffer. This leaves more video memory for textures. The method used to perform hidden-surface removal is hardware-dependent and is transparent to the application.

Z-bufferless HSR is performed if no z-buffer surface is attached to the rendering-target surface and the z-buffer comparison test is enabled (that is, when the state value associated with the **D3DRENDERSTATE_ZENABLE** enumeration constant is set to TRUE).

This flag was introduced in DirectX 5.

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

D3DPBLENDCAPS_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.

D3DPBLENDCAPS_DESTALPHA

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

D3DPBLENDCAPS_DESTCOLOR

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

D3DPBLENDCAPS_INVDESTALPHA

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

D3DPBLENDCAPS_INVDESTCOLOR

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

D3DPBLENDCAPS_INVSRCALPHA

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

D3DPBLENDCAPS_INVSRCCOLOR

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

D3DPBLENDCAPS_ONE

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

D3DPBLENDCAPS_SRCALPHA

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

D3DPBLENDCAPS_SRCALPHASAT

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

D3DPBLENDCAPS_SRCCOLOR

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

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. If this member is zero, the driver does not support alpha tests.

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).

With the monochromatic shade modes, the blue channel of the specular component is interpreted as a white intensity. (This is controlled by the **D3DRENDERSTATE_MONOENABLE** render state.)

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 currently supported.

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 currently supported.

D3DPSHADECAPS_FOGFLAT

D3DPSHADECAPS_FOGGOURAUD

D3DPSHADECAPS_FOGPHONG

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

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 currently supported.

dwTextureCaps

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

D3DPTEXTURECAPS_ALPHA

Supports RGBA textures 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

Supports texture mapping along borders.

D3DPTEXTURECAPS_PERSPECTIVE

Perspective correction is supported.

D3DPTEXTURECAPS_POW2

All nonmipmapped 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 2×2 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 D3DPTFILTERCAPS_MIPLINEAR, but interpolates between the two nearest mipmaps.

D3DPTFILTERCAPS_LINEARMIPNEAREST

The mipmap chosen is the mipmap whose texels most closely match the size of the pixel to be textured. The D3DFILTER_LINEAR method is then used with the texture.

D3DPTFILTERCAPS_MIPLINEAR

Two mipmaps are chosen whose texels most closely match the size of the pixel to be textured. The D3DFILTER_NEAREST method is then used with each texture to produce two values which are then weighted to produce a final texel value.

D3DPTFILTERCAPS_MIPNEAREST

Similar to D3DPTFILTERCAPS_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_ADD

Supports the additive texture-blending mode, in which the Gouraud interpolants are added to the texture lookup with saturation semantics. This capability corresponds to the **D3DTBLEND_ADD** member of the **D3DTEXTUREBLEND** enumerated type.

This flag was introduced in DirectX 5.

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_BORDER

Device supports setting coordinates outside the range [0.0, 1.0] to the border color, as specified by the **D3DRENDERSTATE_BORDERCOLOR** render state. This ability corresponds to the **D3DTADDRESS_BORDER** texture-addressing mode.

This flag was introduced in DirectX 5.

D3DPTADDRESSCAPS_CLAMP

Device can clamp textures to addresses.

D3DPTADDRESSCAPS_INDEPENDENTUV

Device can separate the texture-addressing modes of the U and V coordinates of the texture. This ability corresponds to the **D3DRENDERSTATE_TEXTUREADDRESSU** and **D3DRENDERSTATE_TEXTUREADDRESSV** render-state values.

This flag was introduced in DirectX 5.

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 32×32).

D3DPROCESSVERTICES

The **D3DPROCESSVERTICES** structure 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.

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

Members

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

The **D3DRECT** structure is a rectangle definition.

```
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;
```

Members

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

IDirect3DDevice::Pick, **IDirect3DViewport2::Clear**

D3DSPAN

The **D3DSPAN** structure 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.

```
typedef struct _D3DSPAN {
    WORD wCount;
    WORD wFirst;
} D3DSPAN, *LPD3DSPAN;
```

Members

wCount
: Number of spans.

wFirst
: Index to first vertex.

See Also

D3DOPCODE

D3DSTATE

The **D3DSTATE** structure 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.

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

Members

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

The **D3DSTATS** structure contains statistics used by the **IDirect3DDevice2::GetStats** method.

```
typedef struct _D3DSTATS {
    DWORD dwSize;
    DWORD dwTrianglesDrawn;
    DWORD dwLinesDrawn;
    DWORD dwPointsDrawn;
    DWORD dwSpansDrawn;
    DWORD dwVerticesProcessed;
} D3DSTATS, *LPD3DSTATS;
```

Members

dwSize
: Size, in bytes, of this structure. This member must be initialized before the structure is used.

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

IDirect3DDevice2::GetStats

D3DSTATUS

The **D3DSTATUS** structure 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.

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

Members

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_CLIPINTERSECTIONBACK

Logical **AND** of the clip flags for the vertices compared to the back clipping plane 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_CLIPINTERSECTIONGEN0 through
D3DSTATUS_CLIPINTERSECTIONGEN5

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

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.

Clip Union Flags

D3DSTATUS_CLIPUNIONBACK

Equal to D3DCLIP_BACK.

D3DSTATUS_CLIPUNIONBOTTOM

Equal to D3DCLIP_BOTTOM.

D3DSTATUS_CLIPUNIONFRONT

Equal to D3DCLIP_FRONT.

D3DSTATUS_CLIPUNIONGEN0 through D3DSTATUS_CLIPUNIONGEN5
Equal to D3DCLIP_GEN0 through D3DCLIP_GEN5.

D3DSTATUS_CLIPUNIONLEFT
Equal to D3DCLIP_LEFT.

D3DSTATUS_CLIPUNIONRIGHT
Equal to D3DCLIP_RIGHT.

D3DSTATUS_CLIPUNIONTOP
Equal to D3DCLIP_TOP.

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.

Remarks

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

The **D3DTEXTURELOAD** structure describes operand data for the **D3DOP_TEXTURELOAD** opcode in the **D3DOPCODE** enumerated type.

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

Members

hDestTexture
Handle to the destination texture.

hSrcTexture
Handle to the source texture.

Remarks

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

D3DTLVERTEX

The **D3DTLVERTEX** structure defines a transformed and lit vertex (screen coordinates with color) for the **D3DLIGHTDATA** structure.

```
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;
```

Members

dvSX, **dvSY**, and **dvSZ**
: Values of the **D3DVALUE** type describing a vertex in screen coordinates. The largest allowable value for **dvSZ** is 0.99999, if you want the vertex to be within the range of z-values that are displayed.

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.

Remarks

Direct3D uses the current viewport parameters (the **dwX**, **dwY**, **dwWidth**, and **dwHeight** members of the **D3DVIEWPORT2** structure) to clip **D3DTLVERTEX** vertices. The system always clips z coordinates to [0, 1]. To prevent the system from clipping these vertices, use the D3DDP_DONOTCLIP flag in your call to **IDirect3DDevice2::Begin**.

Prior to DirectX 5, Direct3D did not clip **D3DTLVERTEX** vertices.

See Also

D3DLIGHTDATA, **D3DLVERTEX**, **D3DVERTEX**

D3DTRANSFORMCAPS

The **D3DTRANSFORMCAPS** structure describes the transformation capabilities of a device. This structure is part of the **D3DDEVICEDESC** structure.

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

Members

dwSize
Size, in bytes, of this structure. This member must be initialized before the structure is used.

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

The **D3DTRANSFORMDATA** structure contains information about transformations for the **IDirect3DViewport2::TransformVertices** method.

```
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;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

lpIn
Address of the vertices to be transformed. This should be a **D3DLVERTEX** structure.

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_GEN0 through D3DCLIP_GEN5
Application-defined clipping planes.

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.

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

D3DSTATUS_CLIPINTERSECTIONBACK
Logical **AND** of the clip flags for the vertices compared to the back clipping plane 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_CLIPINTERSECTIONGEN0 through D3DSTATUS_CLIPINTERSECTIONGEN5

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

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.

dwClipUnion

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

D3DSTATUS_CLIPUNIONBACK	Equal to D3DCLIP_BACK.
D3DSTATUS_CLIPUNIONBOTTOM	Equal to D3DCLIP_BOTTOM.
D3DSTATUS_CLIPUNIONFRONT	Equal to D3DCLIP_FRONT.
D3DSTATUS_CLIPUNIONGEN0 through D3DSTATUS_CLIPUNIONGEN5	Equal to D3DCLIP_GEN0 through D3DCLIP_GEN5.
D3DSTATUS_CLIPUNIONLEFT	Equal to D3DCLIP_LEFT.
D3DSTATUS_CLIPUNIONRIGHT	Equal to D3DCLIP_RIGHT.
D3DSTATUS_CLIPUNIONTOP	Equal to D3DCLIP_TOP.

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

Remarks

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

IDirect3DViewport2::TransformVertices

D3DTRIANGLE

The **D3DTRIANGLE** structure 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.

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

Members

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

wFlags
This value can be a combination of the following flags:

Edge flags

These flags describe which edges of the triangle to enable. (This information is useful only in wireframe mode.)

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)

Cull or render the triangles in the strip or fan based on the treatment of this triangle. That is, if this triangle is culled, also cull the specified number of subsequent triangles. If this triangle is rendered, also render the specified number of subsequent triangles.

This length must be greater than zero and less than 30.

Remarks

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

For more information, see Triangle Strips and Fans.

D3DVECTOR

The **D3DVECTOR** structure defines a vector for many Direct3D and Direct3DRM methods and structures.

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

Members

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

See Also

D3DLIGHT2, **D3DLIGHTINGELEMENT**,

D3DVERTEX

The **D3DVERTEX** structure 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.

```
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;
```

Members

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

D3DLVERTEX, **D3DTLVERTEX**, **D3DVALUE**,

D3DVIEWPORT

The **D3DVIEWPORT** structure defines the visible 3-D volume and the 2-D screen area that a 3-D volume projects onto for the **IDirect3DViewport2::GetViewport** and **IDirect3DViewport2::SetViewport** methods.

For the **IDirect3D2** and **IDirect3DDevice2** interfaces, this structure has been superseded by the **D3DVIEWPORT2** structure.

```
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;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before the structure is used.

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 how coordinates are scaled. The relevant coordinates here are the nonhomogeneous coordinates that result from the perspective division that projects the vertices onto the w=1 plane.

dvMaxX, **dvMaxY**, **dvMinZ**, and **dvMaxZ**

Values of the **D3DVALUE** type describing the maximum and minimum nonhomogeneous coordinates of x, y, and z. Again, the relevant coordinates are the nonhomogeneous coordinates that result from the perspective division.

Remarks

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.

See Also

D3DVALUE, **IDirect3DViewport2::GetViewport**, **IDirect3DViewport2::SetViewport**

D3DVIEWPORT2

The **D3DVIEWPORT2** structure defines the visible 3-D volume and the window dimensions that a 3-D volume projects onto. This structure is used by the methods of the **IDirect3D2** and **IDirect3DDevice2** interfaces, and in particular by the **IDirect3DViewport2::GetViewport2** and **IDirect3DViewport2::SetViewport2** methods. This structure was introduced in DirectX 5.

```
typedef struct _D3DVIEWPORT2 {
    DWORD       dwSize;
    DWORD       dwX;
    DWORD       dwY;
    DWORD       dwWidth;
    DWORD       dwHeight;
    D3DVALUE    dvClipX;
    D3DVALUE    dvClipY;
    D3DVALUE    dvClipWidth;
    D3DVALUE    dvClipHeight;
    D3DVALUE    dvMinZ;
    D3DVALUE    dvMaxZ;
} D3DVIEWPORT2, *LPD3DVIEWPORT2;
```

Members

dwSize

Size of this structure, in bytes. This member must be initialized before the structure is used.

dwX and **dwY**
: Coordinates of the top-left corner of the viewport. Unless you want to render to a subset of the surface, these members can be set to 0.

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

dvClipX and **dvClipY**
: Coordinates of the top-left corner of the clipping volume.

  The relevant coordinates here are the nonhomogeneous coordinates that result from the perspective division that projects the vertices onto the w=1 plane.

dvClipWidth and **dvClipHeight**
: Dimensions of the clipping volume projected onto the w=1 plane. Unless you want to render to a subset of the surface, these members can be set to the width and height of the destination surface.

dvMinZ and **dvMaxZ**
: Values of the **D3DVALUE** type describing the maximum and minimum nonhomogeneous z-coordinates resulting from the perspective divide and projected onto the w=1 plane.

Remarks

The coordinates and dimensions of the viewport are given relative to the top left of the device; values increase in the y-direction as you descend the screen.

If you are using **D3DVERTEX** or **D3DLVERTEX** vertices — that is, if Direct3D is performing the transformations — you might want to set the last six members of this structure as follows:

```
float inv_aspect = (float)dwHeight/dwWidth;

dvClipX = -1.0f;
dvClipY = inv_aspect;
dvClipWidth = 2.0f;
dvClipHeight = 2.0f * inv_aspect;
dvMinZ = 0.0f;
dvMaxZ = 1.0f;
```

By taking the aspect ratio into account you are assured that as the surface is resized the angle of the horizontal field of view remains constant. This prevents unexpected distortions when the user pulls the window into an unusual shape. If distortion is not an issue in your application, set *aspect* to 1. Notice that dividing the height by the width produces an inverse aspect ratio; in Direct3D, the aspect ratio is defined by dividing the width by the height.

If you are using **D3DTLVERTEX** vertices — that is, if your application is taking care of the transformations and lighting — you can set up the clip space however is best for your application. If the x- and y-coordinates in your data already match pixels, you could set the last six members of **D3DVIEWPORT2** as follows:

```
dvClipX = 0;
dvClipY = 0;
dvClipWidth = dwWidth;
dvClipHeight = dwHeight;
dvMinZ = 0.0f;
dvMaxZ = 1.0f;
```

Unlike the **D3DVIEWPORT** structure, **D3DVIEWPORT2** specifies the relationship between the size of the viewport and the window.

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

For more information about working with viewports, see Viewports and Transformations.

See Also

D3DVALUE, **IDirect3DViewport2::GetViewport2**, **IDirect3DViewport2::SetViewport2**

Enumerated Types

This section contains information about the following enumerated types used with Direct3D Immediate Mode.

- **D3DANTIALIASMODE**
- **D3DBLEND**
- **D3DCMPFUNC**
- **D3DCOLORMODEL**
- **D3DCULL**
- **D3DFILLMODE**
- **D3DFOGMODE**
- **D3DLIGHTSTATETYPE**
- **D3DLIGHTTYPE**
- **D3DOPCODE**
- **D3DPRIMITIVETYPE**
- **D3DRENDERSTATETYPE**
- **D3DSHADEMODE**
- **D3DTEXTUREADDRESS**
- **D3DTEXTUREBLEND**

- **D3DTEXTUREFILTER**
- **D3DTRANSFORMSTATETYPE**
- **D3DVERTEXTYPE**

D3DANTIALIASMODE

The **D3DANTIALIASMODE** enumerated type defines the supported antialiasing mode for the **D3DRENDERSTATE_ANTIALIAS** value in the **D3DRENDERSTATETYPE** enumerated type. These values define the settings for antialiasing the edges of primitives.

This type was introduced with DirectX 5.

```
typedef enum _D3DANTIALIASMODE {
    D3DANTIALIAS_NONE          = 0,
    D3DANTIALIAS_SORTDEPENDENT = 1,
    D3DANTIALIAS_SORTINDEPENDENT = 2
    D3DANTIALIAS_FORCE_DWORD     = 0x7fffffff,
} D3DANTIALIASMODE;
```

Members

D3DANTIALIAS_NONE
No antialiasing is performed. This is the default setting.

D3DANTIALIAS_SORTDEPENDENT
Antialiasing is dependent on the sort order of the polygons (back-to-front or front-to-back). The application must draw polygons in the right order for antialiasing to occur.

D3DANTIALIAS_SORTINDEPENDENT
Antialiasing is not dependent on the sort order of the polygons.

D3DANTIALIAS_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

D3DBLEND

The **D3DBLEND** enumerated type defines the supported blend mode for the **D3DRENDERSTATE_DESTBLEND** 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*.

```
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_FORCE_DWORD   = 0x7fffffff,
} D3DBLEND;
```

Members

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 (1-R_s, 1-G_s, 1-B_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, 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-A_s, 1-A_s, 1-A_s, 1-A_s), and destination blend factor is (A_s, A_s, A_s, A_s); the destination blend selection is overridden.

D3DBLEND_FORCE_DWORD

Forces this enumerated type to be 32 bits in size.

D3DCMPFUNC

The **D3DCMPFUNC** enumerated type defines the supported compare functions for the **D3DRENDERSTATE_ZFUNC** and **D3DRENDERSTATE_ALPHAFUNC** values of the **D3DRENDERSTATETYPE** enumerated type.

```
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,
    D3DCMP_FORCE_DWORD  = 0x7fffffff,
} D3DCMPFUNC;
```

Members

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.

D3DCMP_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

D3DCOLORMODEL

The **D3DCOLORMODEL** constant defines the color model in which the system will run. A driver can expose either or both flags in the **dcmColorModel** member of the **D3DDEVICEDESC** structure.

```
typedef DWORD D3DCOLORMODEL
```

Values

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.

Remarks

Prior to DirectX 5, these values were part of an enumerated type. This was not correct, because they are bit flags. The enumerated type in earlier versions of DirectX had this syntax:

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

See Also

D3DDEVICEDESC, **D3DFINDDEVICESEARCH**, **D3DLIGHTSTATETYPE**

D3DCULL

The **D3DCULL** enumerated type defines the supported cull modes. These define how back faces are culled when rendering a geometry.

```
typedef enum _D3DCULL {
    D3DCULL_NONE = 1,
    D3DCULL_CW   = 2,
    D3DCULL_CCW  = 3,
    D3DCULL_FORCE_DWORD   = 0x7fffffff,
} D3DCULL;
```

Members

D3DCULL_NONE

Do not cull back faces.

D3DCULL_CW
Cull back faces with clockwise vertices.

D3DCULL_CCW
Cull back faces with counterclockwise vertices.

D3DCULL_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

D3DPRIMCAPS, **D3DRENDERSTATETYPE**

D3DFILLMODE

The **D3DFILLMODE** enumerated type contains constants describing the fill mode. These values are used by the **D3DRENDERSTATE_FILLMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DFILLMODE {
    D3DFILL_POINT     = 1,
    D3DFILL_WIREFRAME = 2,
    D3DFILL_SOLID     = 3
    D3DFILL_FORCE_DWORD   = 0x7fffffff,
} D3DFILLMODE;
```

Members

D3DFILL_POINT
Fill points.

D3DFILL_WIREFRAME
Fill wireframes. This fill mode currently does not work for clipped primitives when you are using the DrawPrimitive methods.

D3DFILL_SOLID
Fill solids.

D3DFILL_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

D3DFOGMODE

The **D3DFOGMODE** enumerated type contains constants describing the fog mode. These values are used by the **D3DRENDERSTATE_FOGTABLEMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DFOGMODE {
    D3DFOG_NONE   = 0,
    D3DFOG_EXP    = 1,
    D3DFOG_EXP2   = 2,
    D3DFOG_LINEAR = 3
    D3DFOG_FORCE_DWORD   = 0x7fffffff,
} D3DFOGMODE;
```

Members

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 currently supported.

D3DFOG_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

Remarks

In monochromatic (ramp) lighting mode, fog works properly only when the fog color is black. (If there is no lighting, any fog color will work, since in this case any fog color is effectively black.)

For more information about fog, see Colors and Fog.

Note

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

The **D3DLIGHTSTATETYPE** enumerated type defines the light state for the **D3DOP_STATELIGHT** opcode. This enumerated type is part of the **D3DSTATE** structure.

```
typedef enum _D3DLIGHTSTATETYPE {
    D3DLIGHTSTATE_MATERIAL    = 1,
    D3DLIGHTSTATE_AMBIENT     = 2,
    D3DLIGHTSTATE_COLORMODEL = 3,
    D3DLIGHTSTATE_FOGMODE     = 4,
    D3DLIGHTSTATE_FOGSTART    = 5,
    D3DLIGHTSTATE_FOGEND      = 6,
    D3DLIGHTSTATE_FOGDENSITY = 7,
    D3DLIGHTSTATE_FORCE_DWORD      = 0x7fffffff,
} D3DLIGHTSTATETYPE;
```

Members

D3DLIGHTSTATE_MATERIAL
Defines the material that is lit and used to compute the final color and intensity values during rasterization. The default value is NULL.

This value must be set when you use textures in ramp mode.

D3DLIGHTSTATE_AMBIENT
Sets the color and intensity of the current ambient light. If an application specifies this value, it should not specify a light as a parameter. 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.

D3DLIGHTSTATE_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

D3DOPCODE and **D3DSTATE**

D3DLIGHTTYPE

The **D3DLIGHTTYPE** enumerated type defines the light type. This enumerated type is part of the **D3DLIGHT2** structure.

```
typedef enum _D3DLIGHTTYPE {
    D3DLIGHT_POINT          = 1,
    D3DLIGHT_SPOT           = 2,
    D3DLIGHT_DIRECTIONAL    = 3,
    D3DLIGHT_PARALLELPOINT = 4,
    D3DLIGHT_FORCE_DWORD    = 0x7fffffff,
} D3DLIGHTTYPE;
```

Members

D3DLIGHT_POINT
: Light is a point source. The light has a position in space and radiates light in all directions.

D3DLIGHT_SPOT
: Light is a spotlight source. This light is something like a point light except that the illumination is limited to a cone. This light type has a direction and several other parameters which determine the shape of the cone it produces. For information about these parameters, see the **D3DLIGHT2** structure.

D3DLIGHT_DIRECTIONAL
: Light is a directional source. This is equivalent to using a point light source at an infinite distance.

D3DLIGHT_PARALLELPOINT
: Light is a parallel point source. This light type acts like a directional light except its direction is the vector going from the light position to the origin of the geometry it is illuminating.

D3DLIGHT_FORCE_DWORD
: Forces this enumerated type to be 32 bits in size.

Remarks

Directional and parallel-point lights are slightly faster than point light sources, but point lights look a little better. Spotlights offer interesting visual effects but are computationally expensive.

D3DOPCODE

The **D3DOPCODE** enumerated type contains the opcodes for execute buffer.

```
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,
    D3DOP_FORCE_DWORD     = 0x7fffffff,
} D3DOPCODE;
```

Members

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.

D3DOP_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

Remarks

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

D3DPRIMITIVETYPE

The **D3DPRIMITIVETYPE** enumerated type lists the primitives supported by DrawPrimitive methods. This type was introduced in DirectX 5.

```
typedef enum _D3DPRIMITIVETYPE {
    D3DPT_POINTLIST     = 1,
    D3DPT_LINELIST      = 2,
    D3DPT_LINESTRIP     = 3,
    D3DPT_TRIANGLELIST  = 4,
    D3DPT_TRIANGLESTRIP = 5,
    D3DPT_TRIANGLEFAN   = 6
    D3DPT_FORCE_DWORD   = 0x7fffffff,
} D3DPRIMITIVETYPE;
```

Members

D3DPT_POINTLIST
Renders the vertices as a collection of isolated points.

D3DPT_LINELIST
Renders the vertices as a list of isolated straight line segments. Calls using this primitive type will fail if the count is less than 2, or is odd.

D3DPT_LINESTRIP
Renders the vertices as a single polyline. Calls using this primitive type will fail if the count is less than 2.

D3DPT_TRIANGLELIST
Renders the specified vertices as a sequence of isolated triangles. Each group of 3 vertices defines a separate triangle. Calls using this primitive type will fail if the count is less than 3, or if not evenly divisible by 3.

Backface culling is affected by the current winding order render state.

D3DPT_TRIANGLESTRIP
Renders the vertices as a triangle strip. Calls using this primitive type will fail if the count is less than 3.

The backface removal flag is automatically flipped on even numbered triangles.

D3DPT_TRIANGLEFAN
Renders the vertices as a triangle fan. Calls using this primitive type will fail if the count is less than 3.

D3DPT_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

IDirect3DDevice2::Begin, **IDirect3DDevice2::BeginIndexed**, **IDirect3DDevice2::DrawIndexedPrimitive**, **IDirect3DDevice2::DrawPrimitive**

D3DRENDERSTATETYPE

The **D3DRENDERSTATETYPE** enumerated type 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.

Values 40 through 49 were introduced with DirectX 5.

```
typedef enum _D3DRENDERSTATETYPE {
    D3DRENDERSTATE_TEXTUREHANDLE     = 1,    // texture handle
    D3DRENDERSTATE_ANTIALIAS         = 2,    // antialiasing mode
```

```
  D3DRENDERSTATE_TEXTUREADDRESS     = 3,    // texture address
  D3DRENDERSTATE_TEXTUREPERSPECTIVE = 4,    // perspective correction
  D3DRENDERSTATE_WRAPU              = 5,    // wrap in u direction
  D3DRENDERSTATE_WRAPV              = 6,    // wrap in v direction
  D3DRENDERSTATE_ZENABLE            = 7,    // enable z test
  D3DRENDERSTATE_FILLMODE           = 8,    // fill mode
  D3DRENDERSTATE_SHADEMODE          = 9,    // shade mode
  D3DRENDERSTATE_LINEPATTERN        = 10,   // line pattern
  D3DRENDERSTATE_MONOENABLE         = 11,   // enable mono rendering
  D3DRENDERSTATE_ROP2               = 12,   // raster operation
  D3DRENDERSTATE_PLANEMASK          = 13,   // physical plane mask
  D3DRENDERSTATE_ZWRITEENABLE       = 14,   // enable z writes
  D3DRENDERSTATE_ALPHATESTENABLE    = 15,   // enable alpha tests
  D3DRENDERSTATE_LASTPIXEL          = 16,   // draw last pixel in a line
  D3DRENDERSTATE_TEXTUREMAG         = 17,   // how textures are magnified
  D3DRENDERSTATE_TEXTUREMIN         = 18,   // how textures are reduced
  D3DRENDERSTATE_SRCBLEND           = 19,   // blend factor for source
  D3DRENDERSTATE_DESTBLEND          = 20,   // blend factor for destination
  D3DRENDERSTATE_TEXTUREMAPBLEND    = 21,   // blend mode for map
  D3DRENDERSTATE_CULLMODE           = 22,   // back-face culling mode
  D3DRENDERSTATE_ZFUNC              = 23,   // z-comparison function
  D3DRENDERSTATE_ALPHAREF           = 24,   // reference alpha value
  D3DRENDERSTATE_ALPHAFUNC          = 25,   // alpha-comparison function
  D3DRENDERSTATE_DITHERENABLE       = 26,   // enable dithering
  D3DRENDERSTATE_BLENDENABLE        = 27,   // replaced by
D3DRENDERSTATE_ALPHABLENDENABLE
  D3DRENDERSTATE_FOGENABLE          = 28,   // enable fog
  D3DRENDERSTATE_SPECULARENABLE     = 29,   // enable specular highlights
  D3DRENDERSTATE_ZVISIBLE           = 30,   // enable z-checking
  D3DRENDERSTATE_SUBPIXEL           = 31,   // enable subpixel correction
  D3DRENDERSTATE_SUBPIXELX          = 32,   // enable x subpixel correction
  D3DRENDERSTATE_STIPPLEDALPHA      = 33,   // enable stippled alpha
  D3DRENDERSTATE_FOGCOLOR           = 34,   // fog color
  D3DRENDERSTATE_FOGTABLEMODE       = 35,   // fog mode
  D3DRENDERSTATE_FOGTABLESTART      = 36,   // fog table start
  D3DRENDERSTATE_FOGTABLEEND        = 37,   // fog table end
  D3DRENDERSTATE_FOGTABLEDENSITY    = 38,   // fog density
  D3DRENDERSTATE_STIPPLEENABLE      = 39,   // enables stippling
  D3DRENDERSTATE_EDGEANTIALIAS      = 40,   // antialias edges
  D3DRENDERSTATE_COLORKEYENABLE     = 41,   // enable color-key transparency
  D3DRENDERSTATE_ALPHABLENDENABLE   = 42,   // enable alpha-blend transparency
  D3DRENDERSTATE_BORDERCOLOR        = 43,   // border color
  D3DRENDERSTATE_TEXTUREADDRESSU    = 44,   // u texture address mode
  D3DRENDERSTATE_TEXTUREADDRESSV    = 45,   // v texture address mode
  D3DRENDERSTATE_MIPMAPLODBIAS      = 46,   // mipmap LOD bias
  D3DRENDERSTATE_ZBIAS              = 47,   // z bias
```

```
    D3DRENDERSTATE_RANGEFOGENABLE     = 48,   // enables range-based fog
    D3DRENDERSTATE_ANISOTROPY         = 49,   // max. anisotropy
    D3DRENDERSTATE_STIPPLEPATTERN00   = 64,   // first line of stipple pattern
      // Stipple patterns 01 through 30 omitted here.
    D3DRENDERSTATE_STIPPLEPATTERN31   = 95,   // last line of stipple pattern
    D3DRENDERSTATE_FORCE_DWORD        = 0x7fffffff,
} D3DRENDERSTATETYPE;
```

Members

D3DRENDERSTATE_TEXTUREHANDLE
: Texture handle. The default value is NULL, which disables texture mapping and reverts to flat or Gouraud shading.

  If the specified texture is in a system memory surface and the driver can only support texturing from display memory surfaces, the call will fail.

  In retail builds the texture handle is not validated.

D3DRENDERSTATE_ANTIALIAS
: One of the members of the **D3DANTIALIASMODE** enumerated type specifying the antialiasing of primitive edges. The default value is **D3DANTIALIAS_NONE**.

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

  Applications that need to specify separate texture-addressing modes for the U and V coordinates of a texture can use the **D3DRENDERSTATE_TEXTUREADDRESSU** and **D3DRENDERSTATE_TEXTUREADDRESSV** render states.

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

  If a square were exactly perpendicular to the viewer, all the points in the square would appear the same. But if the square were tilted with respect to the viewer so that one edge was closer than the other, one side would appear to be longer than the other. Perspective correction ensures that the interpolation of texture coordinates happens correctly in such cases.

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_MONOENABLE
TRUE to enable monochromatic rendering, using a grayscale based on the blue channel of the color rather than full RGB. The default value is FALSE. If the device does not support RGB rendering, the value will be TRUE. Applications can check whether the device supports RGB rendering by using the **dcmColorModel** member of the **D3DDEVICEDESC** structure.

In monochromatic rendering, only the intensity (grayscale) component of the color and specular components are interpolated across the triangle. This means that only one channel (gray) is interpolated across the triangle instead of 3 channels (R,G,B), which is a performance gain for some hardware. This grayscale component is derived from the blue channel of the color and specular components of the triangle.

D3DRENDERSTATE_ROP2
One of the 16 standard Windows 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 the bitwise negation of zero (~0). This physical plane mask can be used to turn off the red bit, the blue bit, and so on.

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. If this state is FALSE, z comparisons are still made according to the render state **D3DRENDERSTATE_ZFUNC** (assuming z-buffering is taking place), but z values are not written to the z-buffer.

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.

The incoming alpha value is compared with the reference alpha value using the comparison function provided by the **D3DRENDERSTATE_ALPHAFUNC** render state. When this mode is enabled, alpha blending occurs only if the test succeeds.

D3DRENDERSTATE_LASTPIXEL

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

D3DRENDERSTATE_TEXTUREMAG
One of the members of the **D3DTEXTUREFILTER** enumerated type. This render state describes how a texture should be filtered when it is being magnified (that is, when a texel must cover more than one pixel). The valid values are **D3DFILTER_NEAREST** (the default) and **D3DFILTER_LINEAR**.

D3DRENDERSTATE_TEXTUREMIN
One of the members of the **D3DTEXTUREFILTER** enumerated type. This render state describes how a texture should be filtered when it is being made smaller (that is, when a pixel contains more than one texel). Any of the members of the **D3DTEXTUREFILTER** enumerated type can be specified for this render state. The default value is **D3DFILTER_NEAREST**.

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

D3DRENDERSTATE_DESTBLEND
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.

The z value of the pixel is compared with the z-buffer value. If the z value of the pixel passes the comparison function, the pixel is written.

The z value is written to the z-buffer only if the render state **D3DRENDERSTATE_ZWRITEENABLE** is TRUE.

Software rasterizers and many hardware accelerators work faster if the z test fails, since there is no need to filter and modulate the texture if the pixel is not going to be rendered.

D3DRENDERSTATE_ALPHAREF
Value specifying a reference alpha value against which pixels are tested when alpha-testing is enabled. This value's type is **D3DFIXED**. It is a 16.16 fixed-point value in the range [0 – 1]. 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
Replaced by the **D3DRENDERSTATE_ALPHABLENDENABLE** render state for DirectX 5.

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

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

Specular highlights are calculated as though every vertex in the object being lit were at the object's origin. This gives the expected results as long as the object is modeled around the origin and the distance from the light to the object is relatively large.

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.

In this mode of operation, the primitives are rendered without writing pixels or updating the z-buffer, and the driver returns TRUE if any of them would be visible. Since no pixels are rendered, this operation is often much faster than it would be if the primitives were naively rendered.

Direct3D's retained mode uses this operation as a quick-reject test: it does the z-visible test on the bounding box of a set of primitives and only renders them if it returns TRUE.

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

Subpixel correction is the ability to draw pixels in precisely their correct locations. In a system that implemented subpixel correction, if a pixel were at position 0.1356, its position would be interpolated from the actual coordinate rather than simply drawn at 0 (using the integer values). Hardware can be non subpixel correct or subpixel correct in x or in both x and y. When interpolating across the x-direction the actual coordinate is used. All hardware should be subpixel correct. Some software rasterizers are not subpixel correct because of the performance loss.

Subpixel correction means that the hardware always pre-steps the interpolant values in the x-direction to the nearest pixel centers and then steps one pixel at a time in the y-direction. For each x span it also pre-steps in the x-direction to the nearest pixel center and then steps in the x-direction one pixel each time. This results in very accurate rendering and eliminates almost all jittering of pixels on triangle edges. Most hardware either doesn't support it (always off) or always supports it (always on).

D3DRENDERSTATE_SUBPIXELX
TRUE to enable subpixel correction in the x direction only. The default value is FALSE.

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

Current software rasterizers ignore this render state. You can use the D3DPSHADECAPS_ALPHAFLATSTIPPLED flag in the **D3DPRIMCAPS** structure to discover whether the current hardware supports this render state.

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
Position in fog table at which fog effects begin for linear fog mode. You specify a position in the fog table with a value between 0.0 and 1.0. This render state enables you to exclude fog effects for positions close to the camera; for example, you could set this value to 0.3 to prevent fog effects for positions between 0.0 and 0.299.

D3DRENDERSTATE_FOGTABLEEND
Position in fog table at which fog effects end for linear fog mode. You specify a position in the fog table with a value between 0.0 and 1.0. This render state enables you to set a position in the fog table at which fog effects will not increase. For example, you could set this value to 0.7 to prevent additional fog effects for positions between 0.701 and 1.0.

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 overrides the current stipple pattern, as specified by the **D3DRENDERSTATE_STIPPLEPATTERN00** through **D3DRENDERSTATE_STIPPLEPATTERN31** render states. When stippled alpha is disabled, the stipple pattern must be returned.

D3DRENDERSTATE_EDGEANTIALIAS
TRUE to antialias lines forming the convex outline of objects. The default value is FALSE. When set to TRUE, only lines should be drawn. The behavior is undefined if triangles or points are drawn when this render state is set. Antialiasing is performed simply by averaging the values of neighboring pixels. Although this is not the best way to perform antialiasing, it can be very efficient; hardware that supports this kind of operation is becoming more common.

Applications should not antialias interior edges of objects. The lines forming the outside edges should be drawn last.

D3DRENDERSTATE_COLORKEYENABLE

TRUE to enable color-keyed transparency. The default value is FALSE. You can use this render state with **D3DRENDERSTATE_ALPHABLENDENABLE** to implement fine blending control.

This render state was introduced in DirectX 5. Applications should check the D3DDEVCAPS_DRAWPRIMTLVERTEX flag in the **D3DDEVICEDESC** structure to find out whether this render state is supported.

D3DRENDERSTATE_ALPHABLENDENABLE

TRUE to enable alpha-blended transparency. The default value is FALSE.

Prior to DirectX 5, this render state was called **D3DRENDERSTATE_BLENDENABLE**. Its name was changed to make its meaning more explicit.

Prior to DirectX 5, the software rasterizers used this render state to toggle both color keying and alpha blending. With DirectX 5, you should use the **D3DRENDERSTATE_COLORKEYENABLE** render state to toggle color keying. (Hardware rasterizers have always used the **D3DRENDERSTATE_BLENDENABLE** render state only for toggling alpha blending.)

The type of alpha blending is determined by the **D3DRENDERSTATE_SRCBLEND** and **D3DRENDERSTATE_DESTBLEND** render states. **D3DRENDERSTATE_ALPHABLENDENABLE**, with **D3DRENDERSTATE_COLORKEYENABLE**, allows fine blending control.

D3DRENDERSTATE_ALPHABLENDENABLE does not affect the texture-blending modes specified by the **D3DTEXTUREBLEND** enumerated type. Texture blending is logically well before the **D3DRENDERSTATE_ALPHABLENDENABLE** part of the pixel pipeline. The only interaction between the two is that the alpha portions remaining in the polygon after the **D3DTEXTUREBLEND** phase may be used in the **D3DRENDERSTATE_ALPHABLENDENABLE** phase to govern interaction with the content in the frame buffer.

Applications should check the D3DDEVCAPS_DRAWPRIMTLVERTEX flag in the **D3DDEVICEDESC** structure to find out whether this render state is supported.

D3DRENDERSTATE_BORDERCOLOR

A **DWORD** value specifying a border color. If the texture addressing mode is specified as **D3DTADDRESS_BORDER** (as set in the **D3DTEXTUREADDRESS** enumerated type), this render state specifies the border color the system uses when it encounters texture coordinates outside the range [0.0, 1.0].

The format of the physical-color information specified by the **DWORD** value depends on the format of the DirectDraw surface.

D3DRENDERSTATE_TEXTUREADDRESSU

One of the members of the **D3DTEXTUREADDRESS** enumerated type. The default value is **D3DTADDRESS_WRAP**. This render state applies only to the U texture coordinate.

This render state, along with **D3DRENDERSTATE_TEXTUREADDRESSV**, allows you to specify separate texture-addressing modes for the U and V coordinates of a texture. Because the **D3DRENDERSTATE_TEXTUREADDRESS** render state applies to both the U and V texture coordinates, it overrides any values set for the **D3DRENDERSTATE_TEXTUREADDRESSU** render state.

D3DRENDERSTATE_TEXTUREADDRESSV

One of the members of the **D3DTEXTUREADDRESS** enumerated type. The default value is **D3DTADDRESS_WRAP**. This render state applies only to the V texture coordinate.

This render state, along with **D3DRENDERSTATE_TEXTUREADDRESSU**, allows you to specify separate texture-addressing modes for the U and V coordinates of a texture. Because the **D3DRENDERSTATE_TEXTUREADDRESS** render state applies to both the U and V texture coordinates, it overrides any values set for the **D3DRENDERSTATE_TEXTUREADDRESSV** render state.

D3DRENDERSTATE_MIPMAPLODBIAS

Floating-point **D3DVALUE** value used to change the level of detail (LOD) bias. This value offsets the value of the mipmap level that is computed by trilinear texturing. It is usually in the range –1.0 to 1.0; the default value is 0.0.

Each unit bias (+/-1.0) biases the selection by exactly one mipmap level. A positive bias will cause the use of larger mipmap levels, resulting in a sharper but more aliased image. A negative bias will cause the use of smaller mipmap levels, resulting in a blurrier image. Applying a negative bias also results in the referencing of a smaller amount of texture data, which can boost performance on some systems.

D3DRENDERSTATE_ZBIAS

An integer value in the range 0 to 16 that causes polygons that are physically coplanar to appear separate. Polygons with a high z-bias value will appear in front of polygons with a low value, without requiring sorting for drawing order. Polygons with a value of 1 appear in front of polygons with a value of 0, and so on. The default value is zero.

D3DRENDERSTATE_RANGEFOGENABLE

TRUE to enable range-based fog. (The default value is FALSE, in which case the system uses depth-based fog.) In range-based fog, the distance of an object from the viewer is used to compute fog effects, not the depth of the object (that is, the z-coordinate) in the scene. In range-based fog, all fog methods work as usual, except that they use range instead of depth in the computations.

Range is the correct factor to use for fog computations, but depth is commonly used instead because range is expensive to compute and depth is generally already available. Using depth to calculate fog has the undesirable effect of having the 'fogginess' of peripheral objects change as the eye is rotated — in this case, the depth changes while the range remains constant.

This render state works only with **D3DVERTEX** vertices. When you specify **D3DLVERTEX** or **D3DTLVERTEX** vertices, the F (fog) component of the RGBF fog value should already be corrected for range.

Since no hardware currently supports per-pixel range-based fog, range correction is calculated at vertices.

D3DRENDERSTATE_ANISOTROPY
Integer value that enables a degree of anisotropic filtering. (This is used for bilinear or trilinear filtering.) The value determines the maximum aspect ratio of the sampling filter kernel. To determine the range of appropriate values, use the D3DPRASTERCAPS_ANISOTROPY flag in the **D3DPRIMCAPS** structure.

Anisotropy is the distortion visible in the texels of a 3-D object whose surface is oriented at an angle with respect to the plane of the screen. The anisotropy is measured as the elongation (length divided by width) of a screen pixel that is inverse-mapped into texture space.

D3DRENDERSTATE_STIPPLEPATTERN00 through **D3DRENDERSTATE_STIPPLEPATTERN31**
Stipple pattern. Each render state applies to a separate line of the stipple pattern. Together, these render states specify a 32x32 stipple pattern.

D3DRENDERSTATE_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

D3DOPCODE, **D3DSTATE**

D3DSHADEMODE

The **D3DSHADEMODE** enumerated type describes the supported shade mode for the **D3DRENDERSTATE_SHADEMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DSHADEMODE {
    D3DSHADE_FLAT          = 1,
    D3DSHADE_GOURAUD       = 2,
    D3DSHADE_PHONG         = 3,
    D3DSHADE_FORCE_DWORD  = 0x7fffffff,
} D3DSHADEMODE;
```

Members

D3DSHADE_FLAT
Flat shade mode. The color and specular component of the first vertex in the triangle are used to determine the color and specular component of the face. These colors remain constant across the triangle; that is, they aren’t interpolated.

D3DSHADE_GOURAUD

Gouraud shade mode. The color and specular components of the face are determined by a linear interpolation between all three of the triangle's vertices.

D3DSHADE_PHONG
Phong shade mode is not currently supported.

D3DSHADE_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

D3DRENDERSTATETYPE

D3DTEXTUREADDRESS

The **D3DTEXTUREADDRESS** enumerated type describes the supported texture addressing modes for the **D3DRENDERSTATE_TEXTUREADDRESS** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DTEXTUREADDRESS {
    D3DTADDRESS_WRAP         = 1,
    D3DTADDRESS_MIRROR       = 2,
    D3DTADDRESS_CLAMP        = 3,
    D3DTADDRESS_BORDER       = 4,
    D3DTADDRESS_FORCE_DWORD  = 0x7fffffff,
} D3DTEXTUREADDRESS;
```

Members

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.

D3DTADDRESS_BORDER
Texture coordinates outside the range [0.0, 1.0] are set to the border color, which is a new render state corresponding to

D3DRENDERSTATE_BORDERCOLOR in the **D3DRENDERSTATETYPE** enumerated type.

This member was introduced in DirectX 5.

D3DTADDRESS_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

Remarks

For more information about using the **D3DRENDERSTATE_WRAPU** and **D3DRENDERSTATE_WRAPV** render states, see Textures.

See Also

D3DRENDERSTATETYPE

D3DTEXTUREBLEND

The **D3DTEXTUREBLEND** enumerated type defines the supported texture-blending modes. This enumerated type is used by the **D3DRENDERSTATE_TEXTUREMAPBLEND** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DTEXTUREBLEND {
    D3DTBLEND_DECAL         = 1,
    D3DTBLEND_MODULATE      = 2,
    D3DTBLEND_DECALALPHA    = 3,
    D3DTBLEND_MODULATEALPHA = 4,
    D3DTBLEND_DECALMASK     = 5,
    D3DTBLEND_MODULATEMASK  = 6,
    D3DTBLEND_COPY          = 7,
    D3DTBLEND_ADD           = 8,
    D3DTBLEND_FORCE_DWORD   = 0x7fffffff,
} D3DTEXTUREBLEND;
```

Members

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.

cPix = cTex
aPix = aTex

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; if the texture does not contain an alpha component, alpha values at the vertices in the source are interpolated between vertices.

*cPix = cSrc * cTex*

aPix = aTex

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:

$$C = (1 - A_t) C_o + A_t C_t$$

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.

*cPix = (cSrc * (10 - aTex)) + (aTex * cTex)*

aPix = aSrc

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.

*cPix = cSrc * cTex*

*aPix = aSrc * aTex*

D3DTBLEND_DECALMASK

Decal-mask texture-blending mode is supported.

cPix = lsb(aTex) ? cTex : cSrc

aPix = aSrc

When the least-significant bit of the texture's alpha component is zero, the effect is as if texturing were disabled.

D3DTBLEND_MODULATEMASK

Modulate-mask texture-blending mode is supported.

*cPix = lsb(aTex) ? cTex * cSrc : cSrc*

aPix = aSrc

When the least-significant bit of the texture's alpha component is zero, the effect is as if texturing were disabled.

D3DTBLEND_COPY

Copy texture-blending mode is supported. This mode is an optimization for software rasterization; for applications using a HAL, it is equivalent to the **D3DTBLEND_DECAL** texture-blending mode.

To use copy mode, textures must use the same pixel format and palette format as the destination surface; otherwise nothing is rendered. Copy mode does no lighting and simply copies texture pixels to the screen. This is often a good technique for prelit textured scenes.

cPix = cTex
aPix = aTex

For more information, see Copy Texture-blending Mode.

D3DTBLEND_ADD

Add the Gouraud interpolants to the texture lookup with saturation semantics (that is, if the color value overflows it is set to the maximum possible value). This member was introduced in DirectX 5.

cPix = cTex + cSrc
aPix = aSrc

D3DTBLEND_FORCE_DWORD

Forces this enumerated type to be 32 bits in size.

Remarks

In the formulas given for the members of this enumerated type, the placeholders have the following meanings:

- *cTex* is the color of the source texel
- *aTex* is the alpha component of the source texel
- *cSrc* is the interpolated color of the source primitive
- *aSrc* is the alpha component of the source primitive
- *cPix* is the new blended color value
- *aPix* is the new blended alpha value

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 preexisting 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

The **D3DTEXTUREFILTER** enumerated type defines the supported texture filter modes used by the **D3DRENDERSTATE_TEXTUREMAG** render state in the **D3DRENDERSTATETYPE** enumerated type.

```
typedef enum _D3DTEXTUREFILTER {
    D3DFILTER_NEAREST          = 1,
    D3DFILTER_LINEAR           = 2,
    D3DFILTER_MIPNEAREST       = 3,
    D3DFILTER_MIPLINEAR        = 4,
    D3DFILTER_LINEARMIPNEAREST = 5,
    D3DFILTER_LINEARMIPLINEAR  = 6,
    D3DFILTER_FORCE_DWORD   = 0x7fffffff,
} D3DTEXTUREFILTER;
```

Members

D3DFILTER_NEAREST
: The texel with coordinates nearest to the desired pixel value is used. This is a point filter with no mipmapping.

  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 2×2 area of texels surrounding the desired pixel is used. This is a bilinear filter with no mipmapping.

  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
: The closest mipmap level is chosen and a point filter is applied.

D3DFILTER_MIPLINEAR
: The closest mipmap level is chosen and a bilinear filter is applied within it.

D3DFILTER_LINEARMIPNEAREST
: The two closest mipmap levels are chosen and then a linear blend is used between point filtered samples of each level.

D3DFILTER_LINEARMIPLINEAR
: The two closest mipmap levels are chosen and then combined using a bilinear filter.

D3DFILTER_FORCE_DWORD
: Forces this enumerated type to be 32 bits in size.

Remarks

All of these filter modes are valid with the **D3DRENDERSTATE_TEXTUREMIN** render state, but only the first two (**D3DFILTER_NEAREST** and **D3DFILTER_LINEAR**) are valid with **D3DRENDERSTATE_TEXTUREMAG**.

D3DTRANSFORMSTATETYPE

The **D3DTRANSFORMSTATETYPE** enumerated type describes the transformation state for the **D3DOP_STATETRANSFORM** opcode in the **D3DOPCODE** enumerated type. This enumerated type is part of the **D3DSTATE** structure.

```
typedef enum _D3DTRANSFORMSTATETYPE {
    D3DTRANSFORMSTATE_WORLD         = 1,
    D3DTRANSFORMSTATE_VIEW          = 2,
    D3DTRANSFORMSTATE_PROJECTION    = 3,
    D3DTRANSFORMSTATE_FORCE_DWORD   = 0x7fffffff,
} D3DTRANSFORMSTATETYPE;
```

Members

D3DTRANSFORMSTATE_WORLD, **D3DTRANSFORMSTATE_VIEW**, and **D3DTRANSFORMSTATE_PROJECTION**
Define the matrices for the world, view, and projection transformations. The default values are NULL (the identity matrices).

D3DTRANSFORMSTATE_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

D3DOPCODE, **D3DRENDERSTATETYPE**

D3DVERTEXTYPE

The **D3DVERTEXTYPE** enumerated type lists the vertex types that are supported by Direct3D.

```
typedef enum _D3DVERTEXTYPE {
    D3DVT_VERTEX        = 1,
    D3DVT_LVERTEX       = 2,
    D3DVT_TLVERTEX      = 3
    D3DVT_FORCE_DWORD   = 0x7fffffff,
};
```

Members

D3DVT_VERTEX
All the vertices in the array are of the **D3DVERTEX** type. This setting will cause transformation, lighting and clipping to be applied to the primitive as it is rendered.

D3DVT_LVERTEX
All the vertices in the array are of the **D3DLVERTEX** type. When used with this option, the primitive will have transformations applied during rendering.

D3DVT_TLVERTEX
All the vertices in the array are of the **D3DTLVERTEX** type. Rasterization only will be applied to this data.

D3DVT_FORCE_DWORD
Forces this enumerated type to be 32 bits in size.

See Also

IDirect3DDevice2::Begin, **IDirect3DDevice2::BeginIndexed**, **IDirect3DDevice2::DrawIndexedPrimitive**, **IDirect3DDevice2::DrawPrimitive**

Other Types

This section contains information about the following Direct3D Immediate Mode types that are neither structures nor enumerated types:

- **D3DCOLOR**
- **D3DVALUE**

D3DCOLOR

The **D3DCOLOR** type is the fundamental Direct3D color type.

```
typedef DWORD D3DCOLOR, D3DCOLOR, *LPD3DCOLOR;
```

See Also

D3DRGB, **D3DRGBA**

D3DVALUE

The **D3DVALUE** type is the fundamental Direct3D fractional data type.

```
typedef float D3DVALUE, *LPD3DVALUE;
```

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all Direct3D Immediate Mode methods. See the individual method descriptions for lists of the values each can return.

D3D_OK

D3DERR_BADMAJORVERSION

D3DERR_BADMINORVERSION

D3DERR_DEVICEAGGREGATED (new for DirectX 5)

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_INITFAILED (new for DirectX 5)

D3DERR_INBEGIN (new for DirectX 5)

D3DERR_INVALID_DEVICE (new for DirectX 5)

D3DERR_INVALIDCURRENTVIEWPORT (new for DirectX 5)

D3DERR_INVALIDPALETTE(new for DirectX 5)

D3DERR_INVALIDPRIMITIVETYPE (new for DirectX 5)

D3DERR_INVALIDRAMPTEXTURE (new for DirectX 5)

D3DERR_INVALIDVERTEXTYPE (new for DirectX 5)

D3DERR_LIGHT_SET_FAILED

D3DERR_LIGHTHASVIEWPORT (new for DirectX 5)

D3DERR_LIGHTNOTINTHISVIEWPORT (new for DirectX 5)

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_NOCURRENTVIEWPORT (new for DirectX 5)

D3DERR_NOTINBEGIN (new for DirectX 5)

D3DERR_NOVIEWPORTS (new for DirectX 5)
D3DERR_SCENE_BEGIN_FAILED
D3DERR_SCENE_END_FAILED
D3DERR_SCENE_IN_SCENE
D3DERR_SCENE_NOT_IN_SCENE
D3DERR_SETVIEWPORTDATA_FAILED
D3DERR_SURFACENOTINVIDMEM (new for DirectX 5)
D3DERR_TEXTURE_BADSIZE (new for DirectX 5)
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
D3DERR_VIEWPORTDATANOTSET (new for DirectX 5)
D3DERR_VIEWPORTHASNODEVICE (new for DirectX 5)
D3DERR_ZBUFF_NEEDS_SYSTEMMEMORY (new for DirectX 5)
D3DERR_ZBUFF_NEEDS_VIDEOMEMORY (new for DirectX 5)