CHAPTER 1

Direct3D Retained-Mode Overview

About Retained Mode

This section describes Direct3D's Retained Mode, Microsoft's solution for real-time 3-D graphics on the personal computer. If you need to create a 3-D environment and manipulate it in real time, you should use Direct3D's Retained-Mode API.

Direct3D is integrated tightly with DirectDraw. A DirectDraw object encapsulates both the DirectDraw and Direct3D states—your application can use the **IDirectDraw::QueryInterface** method to retrieve an *IDirect3D* interface to a DirectDraw object.

If you have written code that uses 3-D graphics before, many of the concepts underlying Retained Mode will be familiar to you. If, however, you are new to 3-D programming, you should pay close attention to *Direct3D Retained-Mode Architecture*, and you should read *Getting Started*. Whether you are new to 3-D programming or just beginning, you should look carefully at the sample code included with this SDK; it illustrates how to put Retained Mode to work in real-world applications.

This section is an introduction to 3-D programming. It describes Microsoft's 3D-graphics solutions and some of the technical background you need to manipulate points in three dimensions. It is not an introduction to programming with Direct3D's Retained Mode; for this information, see *Direct3D Retained-Mode Tutorial*.

Getting Started

The following sections describe some of the technical concepts you need to understand before you write programs that incorporate 3-D graphics. In these sections, you will find a general discussion of coordinate systems and transformations. This is not a discussion of broad architectural details, such as

setting up models, lights, and viewing parameters. For more information about these topics, see *Direct3D Retained-Mode Architecture*.

If you are already experienced in producing 3-D graphics, simply scan the following sections for information that is unique to Direct3D Retained Mode.

3D Coordinate Systems

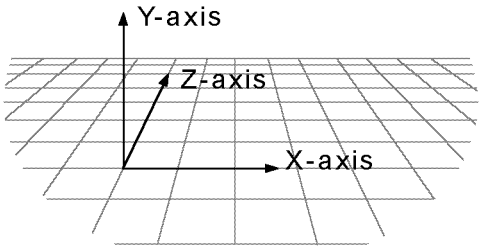
There are two varieties of Cartesian coordinate systems in 3-D graphics: left-handed and right-handed. In both coordinate systems, the positive x-axis points to the right and the positive y-axis points up. You can remember which direction the positive z-axis points by pointing the fingers of either your left or right hand in the positive x direction and curling them into the positive y direction. The direction your thumb points, either toward or away from you, is the direction the positive z-axis points for that coordinate system.

This section describes the Direct3D coordinate system and coordinate types that your application can use.

- *Direct3D's Coordinate System*
- *U- and V-Coordinates*

Direct3D's Coordinate System

Direct3D uses the left-handed coordinate system. This means the positive z-axis points away from the viewer, as shown in the following illustration:



In a left-handed coordinate system, rotations occur clockwise around any axis that is pointed at the viewer.

If you need to work in a right-handed coordinate system—for example, if you are porting an application that relies on right-handedness—you can do so by making two simple changes to the data passed to Direct3D.

- Flip the order of triangle vertices so that the system traverses them clockwise from the front. In other words, if the vertices are v0, v1, v2, pass them to Direct3D as v0, v2, v1.
- Scale the projection matrix by -1 in the z direction. To do this, flip the signs of the _13, _23, _33, and _43 members of the **D3DMATRIX** structure.

U- and V-Coordinates

Direct3D also uses *texture coordinates*. These coordinates (u and v) are used when mapping textures onto an object. The v-vector describes the direction or orientation of the texture and lies along the z-axis. The u-vector (or the *up* vector) typically lies along the y-axis, with its origin at [0,0,0]. For more information about u- and v-coordinates, see *IDirect3DRMWrap Interface*.

3D Transformations

In programs that work with 3-D graphics, you can use geometrical transformations to:

- Express the location of an object relative to another object.
- Rotate, shear, and size objects.
- Change viewing positions, directions, and perspective.

You can transform any point into another point by using a 4×4 matrix. In the following example, a matrix is used to reinterpret the point (x, y, z), producing the new point (x', y', z'):

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

You perform the following operations on (x, y, z) and the matrix to produce the point (x', y', z'):

$$x' = (M_{11} \times x) + (M_{21} \times y) + (M_{31} \times z) + (M_{41} \times 1)$$
$$y' = (M_{12} \times x) + (M_{22} \times y) + (M_{32} \times z) + (M_{42} \times 1)$$
$$z' = (M_{13} \times x) + (M_{23} \times y) + (M_{33} \times z) + (M_{43} \times 1)$$

The most common transformations are translation, rotation, and scaling. You can combine the matrices that produce these effects into a single matrix to calculate

several transformations at once. For example, you can build a single matrix to translate and rotate a series of points.

Matrices are specified in row order. For example, the following matrix could be represented by an array:

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & t & 0 \\ 0 & 0 & s & v \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The array for this matrix would look like the following:

```
D3DMATRIX scale = {
    D3DVAL(s),    0,            0,            0,
    0,            D3DVAL(s),    D3DVAL(t),    0,
    0,            0,            D3DVAL(s),    D3DVAL(v),
    0,            0,            0,            D3DVAL(1)
};
```

This section describes the 3-D transformations available to your applications through Direct3D.

- *Translation*
- *Rotation*
- *Scaling*

Other parts of this documentation also discuss transformations. You can find a general discussion of transformations in the section devoted to viewports in Retained Mode, *Transformations*. For a discussion of transformations in frames, see *Transformations*. Although each of these sections discusses Retained-Mode API, the architecture and mathematics of the transformations apply to both Retained Mode and Immediate Mode.

Translation

The following transformation translates the point (x, y, z) to a new point (x', y', z'):

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Rotation

The transformations described in this section are for left-handed coordinate systems, and so may be different from transformation matrices you have seen elsewhere.

The following transformation rotates the point (x, y, z) around the x-axis, producing a new point (x', y', z'):

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following transformation rotates the point around the y-axis:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following transformation rotates the point around the z-axis:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that in these example matrices, the Greek letter theta stands for the angle of rotation, specified in radians. Angles are measured clockwise when looking along the rotation axis toward the origin.

Scaling

The following transformation scales the point (x, y, z) by arbitrary values in the x-, y-, and z-directions to a new point (x', y', z'):

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Polygons

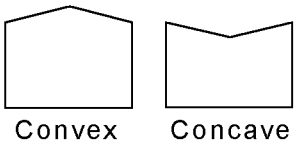
Three-dimensional objects in Direct3D are made up of meshes. A mesh is a set of faces, each of which is described by a simple polygon. The fundamental polygon type is the triangle. Although Retained-Mode applications can specify polygons with more than three vertices, the system translates these into triangles before the objects are rendered. Immediate-Mode applications must use triangles.

This section describes how your applications can use Direct3D polygons.

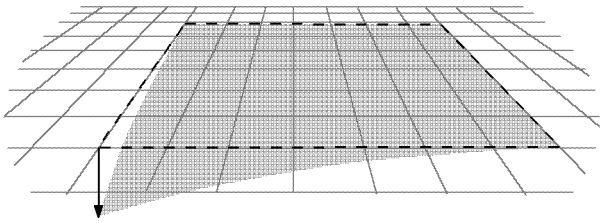
- *Geometry Requirements*
- *Face and Vertex Normals*
- *Shade Modes*
- *Triangle Interpolants*

Geometry Requirements

Triangles are the preferred polygon type because they are always convex and they are always planar, two conditions that are required of polygons by the renderer. A polygon is convex if a line drawn between any two points of the polygon is also inside the polygon.

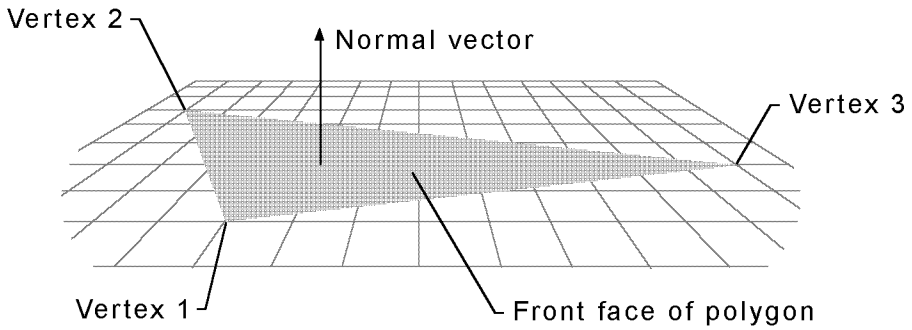


The three vertices of a triangle always describe a plane, but it is easy to accidentally create a nonplanar polygon by adding another vertex.

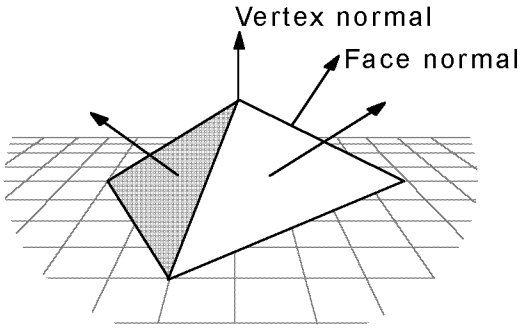


Face and Vertex Normals

Each face in a mesh has a perpendicular face *normal vector* whose direction is determined by the order in which the vertices are defined and by whether the coordinate system is right- or left-handed. If the normal vector of a face is oriented toward the viewer, that side of the face is its front. In Direct3D, only the front side of a face is visible, and a front face is one in which vertices are defined in clockwise order.



Direct3D applications do not need to specify face normals; the system calculates them automatically when they are needed. The system uses face normals in the flat shade mode. For Phong and Gouraud shade modes, and for controlling lighting and texturing effects, the system uses vertex normals.



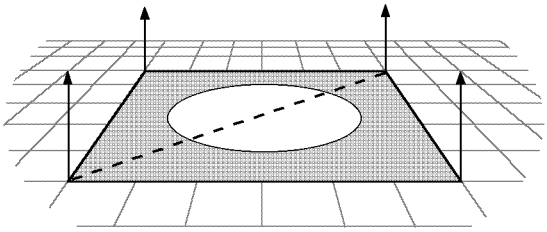
Shade Modes

In the flat shade mode, the system duplicates the color of one vertex across all the other faces of the primitive. In the Gouraud and Phong shade modes, vertex normals are used to give a smooth look to a polygonal object. In Gouraud shading, the color and intensity of adjacent vertices is interpolated across the

space that separates them. In Phong shading, the system calculates the appropriate shade value for each pixel on a face.

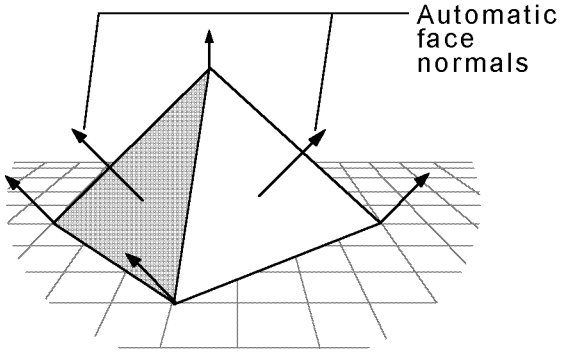
Note Phong shading is not currently supported.

Most applications use Gouraud shading because it allows objects to appear smooth and is computationally efficient. Gouraud shading can miss details that Phong shading will not, however. For example, Gouraud and Phong shading would produce very different results in the case shown by the following illustration, in which a spotlight is completely contained within a face.

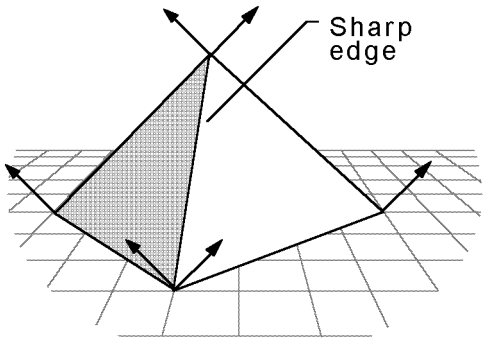


In this case, the Phong shade mode would calculate the value for each pixel and display the spotlight. The Gouraud shade mode, which interpolates between vertices, would miss the spotlight altogether; the face would be rendered as though the spotlight did not exist.

In the flat shade mode, the following pyramid would be displayed with a sharp edge between adjoining faces; the system would generate automatic face normals. In the Gouraud or Phong shade modes, however, shading values would be interpolated across the edge, and the final appearance would be of a curved surface.



If you want to use the Gouraud or Phong shade mode to display curved surfaces and you also want to include some objects with sharp edges, your application would need to duplicate the vertex normals at any intersection of faces where a sharp edge was required, as shown in the following illustration.



In addition to allowing a single object to have both curved and flat surfaces, the Gouraud shade mode lights flat surfaces more realistically than the flat shade mode. A face in the flat shade mode is a uniform color, but Gouraud shading allows light to fall across a face correctly; this effect is particularly obvious if there is a nearby point source. Gouraud shading is the preferred shade mode for most Direct3D applications.

Triangle Interpolants

The system interpolates the characteristics of a triangle's vertices across the triangle when it renders a face. The following are the triangle interpolants:

- Color
- Specular
- Fog
- Alpha

All of the triangle interpolants are modified by the current shade mode:

Flat	No interpolation is done. Instead, the color of the first vertex in the triangle is applied across the entire face.
Gouraud	Linear interpolation is performed between all three vertices.
Phong	Vertex parameters are reevaluated for each pixel in the face, using the current lighting. The Phong shade mode is not currently supported.

The color and specular interpolants are treated differently, depending on the color model. In the RGB color model (**D3DCOLOR_RGB**), the system uses the red, green, and blue color components in the interpolation. In the monochromatic model (**D3DCOLOR_MONO**), the system uses only the blue component of the vertex color.

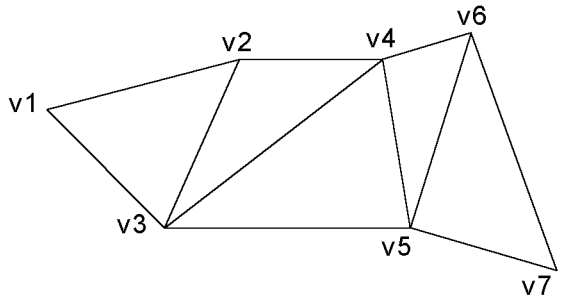
For example, if the red component of the color of vertex 1 were 0.8 and the red component of vertex 2 were 0.4, in the Gouraud shade mode and RGB color model the system would use interpolation to assign a red component of 0.6 to the pixel at the midpoint of the line between these vertices.

The alpha component of a color is treated as a separate interpolant because device drivers can implement transparency in two different ways: by using texture blending or by using stippling.

An application can use the **dwShadeCaps** member of the **D3DPRIMCAPS** structure to determine what forms of interpolation the current device driver supports.

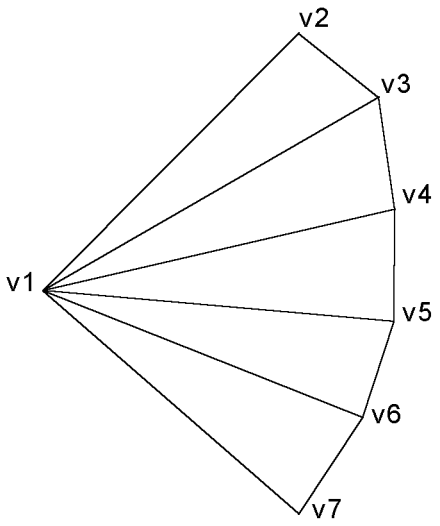
Triangle Strips and Fans

You can use triangle strips and triangle fans to specify an entire surface without having to provide all three vertices for each of the triangles. For example, only seven vertices are required to define the following triangle strip.



The system uses vertices v1, v2, and v3 to draw the first triangle, v2, v4, and v3 to draw the second triangle, v3, v4, and v5 to draw the third, v4, v6, and v5 to draw the fourth, and so on. Notice that the vertices of the second and fourth triangles are out of order; this is required to make sure that all of the triangles are drawn in a clockwise orientation.

A triangle fan is similar to a triangle strip, except that all of the triangles share one vertex.



The system uses vertices v1, v2, and v3 to draw the first triangle, v3, v4, and v1 to draw the second triangle, v1, v4, and v5 to draw the third triangle, and so on.

You can use the **wFlags** member of the **D3DTRIANGLE** structure to specify the flags that build triangle strips and fans.

Vectors, Vertices, and Quaternions

Throughout Direct3D, vertices describe position and orientation. Each vertex in a primitive is described by a vector that gives its position, a normal vector that gives its orientation, texture coordinates, and a color. (In Retained Mode, the **D3DRMVERTEX** structure contains these values.)

Quaternions add a fourth element to the [*x, y, z*] values that define a vector. Quaternions are an alternative to the matrix methods that are typically used for 3-D rotations. A quaternion represents an axis in 3-D space and a rotation around that axis. For example, a quaternion could represent a (1,1,2) axis and a rotation of 1 radian. Quaternions carry valuable information, but their true power comes from the two operations that you can perform on them: *composition* and *interpolation*.

Performing composition on quaternions is similar to combining them. The composition of two quaternions is notated as follows:

$$Q = q_1 \circ q_2$$

The composition of two quaternions applied to a geometry means "rotate the geometry around $axis_2$ by $rotation_2$, then rotate it around $axis_1$ by $rotation_1$." In this

case, Q represents a rotation around a single axis that is the result of applying q_2, then q_1 to the geometry.

Using quaternion interpolation, an application can calculate a smooth and reasonable path from one axis and orientation to another. Therefore, interpolation between q_1 and q_2 provides a simple way to animate from one orientation to another.

When you use composition and interpolation together, they provide you with a simple way to manipulate a geometry in a manner that appears complex. For example, imagine that you have a geometry that you want to rotate to a given orientation. You know that you want to rotate it r_2 degrees around $axis_2$, then rotate it r_1 degrees around $axis_1$, but you don't know the final quaternion. By using composition, you could combine the two rotations on the geometry to get a single quaternion that is the result. Then, you could interpolate from the original to the composed quaternion to achieve a smooth transition from one to the other.

Direct3D's Retained Mode includes some functions that help you work with quaternions. For example, the **D3DRMQuaternionFromRotation** function adds a rotation value to a vector that defines an axis of rotation and returns the result in a quaternion defined by a **D3DRMQUATERNION** structure. Additionally, the **D3DRMQuaternionMultiply** function composes quaternions and the **D3DRMQuaternionSlerp** performs spherical linear interpolation between two quaternions.

Retained-Mode applications can use the following functions to simplify the task of working with vectors and quaternions:

D3DRMQuaternionFromRotation

D3DRMQuaternionMultiply

D3DRMQuaternionSlerp

D3DRMVectorAdd

D3DRMVectorCrossProduct

D3DRMVectorDotProduct

D3DRMVectorModulus

D3DRMVectorNormalize

D3DRMVectorRandom

D3DRMVectorReflect

D3DRMVectorRotate

D3DRMVectorScale

D3DRMVectorSubtract

Floating-point Precision

Direct3D, like the rest of the DirectX architecture, uses a floating-point precision of 53 bits. If your application needs to change this precision, it must change it back to 53 when the calculations are finished. Otherwise, system components that depend on the default value will stop working.

Direct3D Retained-Mode Architecture

All access to Direct3D Retained Mode is through a small set of objects. The following table lists these objects and a brief description of each:

Object	Description
Direct3DRMAnimation	This object defines how a transformation will be modified, often in reference to a Direct3DRMFrame or Direct3DRMFrame2 object. You can use it to animate the position, orientation, and scaling of Direct3DRMVisual, Direct3DRMLight, and Direct3DRMViewport objects.
Direct3DRMAnimationSet	This object allows Direct3DRMAnimation objects to be grouped together.
Direct3DRMDevice	This object represents the visual display destination for the renderer.
Direct3DRMDevice2	This object represents the visual display destination for the renderer. Same as the Direct3DRMDevice object but with enhanced control of transparency.
Direct3DRMFace	This object represents a single polygon in a mesh.
Direct3DRMFrame	This object positions objects within a scene and defines the positions and orientations of visual objects.
Direct3DRMFrame2	Extends the **Direct3DRMFrame** object by enabling access to the frame axes, bounding boxes, and materials. Also supports ray picking.
Direct3DRMInterpolator	This object stores actions and applies the actions to objects with automatic calculation of in-between values.
Direct3DRMLight	This object defines one of five types of lights that are used to illuminate the visual objects in a scene.

Direct3DRMMaterial	This object defines how a surface reflects light.
Direct3DRMMesh	This object consists of a set of polygonal faces. You can use this object to manipulate groups of faces and vertices.
Direct3DRMMeshBuilder	This object allows you to work with individual vertices and faces in a mesh.
Direct3DRMMeshBuilder2	This object allows you to work with individual vertices and faces in a mesh. Same as the **Direct3DRMMeshBuilder** object but with than the **Direct3DRMMeshBuilder** object.
Direct3DRMObject	This object is a base class used by all other Direct3D Retained-Mode objects; it has characteristics that are common to all objects.
Direct3DRMPickedArray	This object identifies a visual object that corresponds to a given 2-D point.
Direct3DRMPicked2Array	This object identifies a visual object that corresponds to a given ray intersection.
Direct3DRMProgressiveMesh	This object consists of a coarse base mesh together with records describing how to incrementally refine the mesh. This allows a generalized level of detail to be set on the mesh as well as progressive download of the mesh from a remote source.
Direct3DRMShadow	This object defines a shadow.
Direct3DRMTexture	This object is a rectangular array of colored pixels.
Direct3DRMTexture2	This object is a rectangular array of colored pixels. Same as the **Direct3DRMTexture** object except that resources can be loaded from files other than the currently executing file, textures can be created from images in memory, and you can generate MIP maps.
Direct3DRMUserVisual	This object is defined by an application to provide functionality not otherwise available in the system.
Direct3DRMViewport	This object defines how the 3-D scene is rendered into a 2-D window.
Direct3DRMVisual	This object is anything that can be rendered in a scene. Visual objects need not be visible; for example, a frame can

	be added as a visual object.
Direct3DRMWrap	This object calculates texture coordinates for a face or mesh.

Many objects can be grouped into arrays, called *array objects*. Array objects make it simpler to apply operations to the entire group. The COM interfaces that allow you to work with array objects contain the **GetElement** and **GetSize** methods. These methods retrieve a pointer to an element in the array and the size of the array, respectively. For more information about array interfaces, see *IDirect3DRM Array Interfaces*.

Objects and Interfaces

Calling the *IObjectName*::**QueryInterface** method retrieves a valid interface pointer only if the object supports that interface; therefore, you could call the **IDirect3DRMDevice::QueryInterface** method to retrieve the **IDirect3DRMWinDevice** interface, but not to retrieve the **IDirect3DRMVisual** interface.

Object name	**Supported interfaces**
Direct3DRMAnimation	*IDirect3DRMAnimation*
Direct3DRMAnimationSet	*IDirect3DRMAnimationSet*
Direct3DRMDevice	*IDirect3DRMDevice*, *IDirect3DRMWinDevice*
Direct3DRMDevice2	*IDirect3DRMDevice2*, *IDirect3DRMWinDevice*
Direct3DRMFace	*IDirect3DRMFace*
Direct3DRMFrame	*IDirect3DRMFrame*, **IDirect3DRMVisual**
Direct3DRMFrame2	*IDirect3DRMFrame2*, **IDirect3DRMVisual**
Direct3DRMInterpolator	*IDirect3DRInterpolator*
Direct3DRMLight	*IDirect3DRMLight*
Direct3DRMMaterial	*IDirect3DRMMaterial*
Direct3DRMMesh	*IDirect3DRMMesh*, **IDirect3DRMVisual**
Direct3DRMProgressiveMesh	*IDirect3DRMProgressiveMesh*, **IDirect3DRMVisual**
Direct3DRMMeshBuilder	*IDirect3DRMMeshBuilder*, **IDirect3DRMVisual**
Direct3DRMMeshBuilder2	*IDirect3DRMMeshBuilder2*, **IDirect3DRMVisual**
Direct3DRMShadow	*IDirect3DRMShadow*, **IDirect3DRMVisual**
Direct3DRMTexture	*IDirect3DRMTexture*, *IDirect3DRMTexture2*, **IDirect3DRMVisual**
Direct3DRMUserVisual	*IDirect3DRMUserVisual*, **IDirect3DRMVisual**
Direct3DRMViewport	*IDirect3DRMViewport*

Direct3DRMWrap	*IDirect3DRMWrap*

The following example illustrates how to create two interfaces to a single Direct3DRMDevice object. The **IDirect3DRM::CreateObject** method creates an uninitialized Direct3DRMDevice object. The **IDirect3DRMDevice::InitFromClipper** method initializes the object. The call to the **IDirect3DRMDevice::QueryInterface** method creates a second interface to the Direct3DRMDevice object—an *IDirect3DRMWinDevice* interface the application will use to handle WM_PAINT and WM_ACTIVATE messages.

```
d3drmapi->CreateObject(CLSID_CDirect3DRMDevice, NULL,
    IID_IDirect3DRMDevice,(LPVOID FAR*)&dev1);
dev1->InitFromClipper(lpDDClipper, IID_IDirect3DRMDevice,
    r.right, r.bottom);
dev1->QueryInterface(IID_IDirect3DRMWinDevice, (LPVOID*) &dev2);
```

To determine if two interfaces refer to the same object, call the **QueryInterface** method of each interface and compare the values of the pointers they return. If the pointer values are the same, the interfaces refer to the same object.

All Direct3D Retained-Mode objects support the **IDirect3DRMObject** and *IUnknown* interfaces in addition to the interfaces in the preceding list. Array objects are not derived from **IDirect3DRMObject**. Array objects have no class identifiers (CLSIDs) because they are not needed. Applications cannot create array objects in a call to the **IDirect3DRM::CreateObject** method; instead, they should use the creation methods listed below for each interface:

Array interface	**Creation method**
IDirect3DRMDeviceArray	**IDirect3DRM::GetDevices**
IDirect3DRMFaceArray	**IDirect3DRMMeshBuilder::GetFaces** or **IDirect3DRMMeshBuilder2::GetFaces**
IDirect3DRMFrameArray	**IDirect3DRMPickedArray::GetPick**, **IDirect3DRMPicked2Array::GetPick**, or **IDirect3DRMFrame::GetChildren**
IDirect3DRMLightArray	**IDirect3DRMFrame::GetLights**
IDirect3DRMObjectArray	**IDirect3DRMInterpolator::GetAttachedObjects**
IDirect3DRMPickedArray	**IDirect3DRMViewport::Pick**
IDirect3DRMPicked2Array	**IDirect3DRMFrame2::RayPick**

IDirect3DRMViewportArray	**IDirect3DRM::CreateFrame**
IDirect3DRMVisualArray	**IDirect3DRMFrame::GetVisuals**

Objects and Reference Counting

Whenever an object is created, its reference count is increased. Each time an application creates a child of an object or a method returns a pointer to an object, the system increases the reference count for that object. The object is not deleted until its reference count reaches zero.

Applications need to keep track of the reference count for a single object only: the root of the scene. The system keeps track of the other reference counts automatically. Applications should be able to simply release the scene, the viewport, and the device when they clean up before exiting. (When your application releases the viewport, the system automatically takes care of the camera's references.) Theoretically, an application could release a viewport without releasing the device (if it needed to add a new viewport to the device, for example), but whenever an application releases a device, it should release the viewport as well.

The reference count of a child or visual object increases whenever it is added to a frame. When you use the **IDirect3DRMFrame::AddChild** method to move a child from one parent to another, the system handles the reference counting automatically.

After your application loads a visual object into a scene, the scene handles the reference counting for the visual object. The application no longer needs the visual object and can release it.

Creating and applying a wrap does not increase the reference count of any objects, because wrapping is really just a convenient method of calculating texture coordinates.

Z Buffers in Retained-Mode

The order in the z buffer determines the order in which overlays clip each other. Overlays are assumed to be on top of all other screen components. Overlays that do not have a specified z-order behave in unpredictable ways when overlaying the same area on the primary surface. Direct3D Retained-Mode does not sort overlays if you do not have a z buffer. Overlays without a specified z-order are assumed to have a z-order of 0, and will appear in the order they are rendered. The possible z-order of overlays ranges from 0, which is just on top of the primary surface, to 4 billion, which is as close to the viewer as possible. An overlay with a z-order of 2

would obscure an overlay with a z-order of 1. No two overlays can have the same z-order.

IDirect3DRM and IDirect3DRM2 Interfaces

Applications use the methods of the **IDirect3DRM** interface to create Direct3DRM objects and work with system-level variables. For a reference to the methods of this interface, see *IDirect3DRM* or *IDirect3DRM2*.

Applications use the methods of the **IDirect3DRMDevice** and **IDirect3DRMDevice2** interfaces to interact with the output device. An **IDirect3DRMDevice** created from the **IDirect3DRM** interface works with an **IDirect3DDevice** Immediate-Mode device. An **IDirect3DRMDevice2** created from the **IDirect3DRM2** interface, or initialized by the **IDirect3DRMDevice2::InitFromClipper**, **IDirect3DRMDevice::InitFromD3D**, or **IDirect3DRMDevice2::InitFromSurface** method works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRM2 supports all the methods in **IDirect3DRM**. An additional method is included **IDirect3DRM2::CreateProgressiveMesh**. The **IDirect3DRM2::CreateDeviceFromSurface** methods, **IDirect3DRM2::CreateDeviceFromD3D**, and **IDirect3DRM2::CreateDeviceFromClipper** all create a DIRECT3DRMDEVICE2 object. The **IDirect3DRM2::CreateViewport** method creates a viewport on a DIRECT3DRMDEVICE2 object. The **IDirect3DRM2::LoadTexture** and **IDirect3DRM2::LoadTextureFromResource** methods load a DIRECT3DRMTEXTURE2 object.

The **IDirect3DRM** COM interface is created by calling the **Direct3DRMCreate** function. To access the **IDirect3DRM2** COM interface, create an **IDirect3DRM** object with **Direct3DRMCreate**, then query for **IDirect3DRM2** from **IDirect3DRM**.

The methods of the **IDirect3DRM** and **IDirect3DRM2** interfaces create the following objects:

- Animations and animation sets
- Devices
- Faces
- Frames
- Generic uninitialized objects
- Lights

- Materials
- Meshes and mesh builders
- Shadows
- Textures
- UserVisuals
- Viewports
- Wraps

In addition, the **IDirect3DRM2::CreateProgressiveMesh** creates a DIRECT3DRMPROGRESSIVEMESH object.

IDirect3DRMAnimation and IDirect3DRMAnimationSet Interfaces

An animation in Retained Mode is defined by a set of *keys*. A key is a time value associated with a scaling operation, an orientation, or a position. A Direct3DRMAnimation object defines how a transformation is modified according to the time value. The animation can be set to operate on a Direct3DRMFrame object, so it could be used to animate the position, orientation, and scaling of Direct3DRMVisual, Direct3DRMLight, and Direct3DRMViewport objects.

The **IDirect3DRMAnimation::AddPositionKey**, **IDirect3DRMAnimation::AddRotateKey**, and **IDirect3DRMAnimation::AddScaleKey** methods each specify a time value whose units are arbitrary. If an application adds a position key with a time value of 99, for example, a new position key with a time value of 49 would occur exactly halfway between the (zero-based) beginning of the animation and the first position key.

The animation is driven by calling the **IDirect3DRMAnimation::SetTime** method. This sets the visual object's transformation to the interpolated position, orientation, and scale of the nearby keys in the animation. As with the methods that add animation keys, the time value for **IDirect3DRMAnimation::SetTime** is an arbitrary value, based on the positions of keys the application has already added.

A Direct3DRMAnimationSet object allows Direct3DRMAnimation objects to be grouped together. This allows all the animations in an animation set to share the same time parameter, simplifying the playback of complex articulated animation sequences. An application can add an animation to an animation set by using the **IDirect3DRMAnimationSet::AddAnimation** method, and it can remove one by using the **IDirect3DRMAnimationSet::DeleteAnimation** method. Animation sets are driven by calling the **IDirect3DRMAnimationSet::SetTime** method.

For related information, see the *IDirect3DRMAnimation* and *IDirect3DRMAnimationSet* interfaces.

IDirect3DRMDevice, IDirect3DRMDevice2, and IDirect3DRMDeviceArray Interfaces

All forms of rendered output must be associated with an output device. The device object represents the visual display destination for the renderer.

The renderer's behavior depends on the type of output device that is specified. You can define multiple viewports on a device, allowing different aspects of the scene to be viewed simultaneously. You can also specify any number of devices, allowing multiple destination devices for the same scene.

Retained Mode supports devices that render directly to the screen, to windows, or into application memory.

While an **IDirect3DRMDevice** interface, when created from the **IDirect3DRM** interface, works with an **IDirect3DDevice** Immediate-Mode device, an **IDirect3DRMDevice2** interface, when created from the **IDirect3DRM2** interface or initialized by the **IDirect3DRMDevice2::InitFromClipper**, **IDirect3DRMDevice2::InitFromD3D2**, or **IDirect3DRMDevice2::InitFromSurface** method, works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

The **IDirect3DRMDevice2::InitFromClipper** and **IDirect3DRMDevice2::InitFromSurface** methods use the **IDirect3DRM2::CreateDevice** method to create an **IDirect3DRMDevice2** object. The **IDirect3DRMDevice2::InitFromD3D2** method uses an **IDirect3D2** Immediate-Mode object and an **IDirect3DDevice2** Immediate-Mode device to initialize an **IDirect3DDevice2** Retained-Mode device.

You can still query back and forth between the **IDirect3DRMDevice** and **IDirect3DRMDevice2** interfaces. The main difference is in how the underlying Immediate-Mode device is created.

The **IDirect3DRMDevice2** interface contains all the methods found in the **IDirect3DRMDevice** interface, plus two additional ones that allow you to control transparency, **IDirect3DRMDevice2::GetRenderMode** and **IDirect3DRMDevice2::SetRenderMode**, and one additional initialization method **IDirect3DRMDevice2::InitFromSurface**.

For related information, see *IDirect3DRMDevice* and *IDirect3DRMDevice2*.

This section describes the options available to display Direct3D images to output devices.

- *Quality*
- *Color Models*
- *Window Management*

Quality

The device allows the scene and its component parts to be rendered with various degrees of realism. The device rendering quality is the maximum quality at which rendering can take place on the rendering surface of that device. Mesh, progressive mesh, and mesh builder objects can also have a specified rendering quality.

A device's or object's quality has three components: shade mode (flat or gouraud, phong is not yet implemented and will default to gouraud shading), lighting type (on or off), and fill mode (point, wireframe or solid).

You can set the quality of a device with **IDirect3DRMDevice::SetQuality** and **IDirect3DRMDevice2::SetQuality** methods. By default, the device quality is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMProgressiveMesh, Direct3DRMMeshBuilder, or Direct3DRMMeshBuilder2 object with their respective **SetQuality** methods, **IDirect3DRMProgressiveMesh::SetQuality**, **IDirect3DRMMeshBuilder::SetQuality**, and **IDirect3DRMMeshBuilder2::SetQuality**. By default, the quality of these objects is D3DRMRENDER_GOURAUD (gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

Color Models

Retained Mode supports two color models: an RGB model and a monochromatic (or ramp) model. To retrieve the color model, an application can use the **IDirect3DRMDevice::GetColorModel** method.

The RGB model treats color as a combination of red, green, and blue light, and it supports multiple light sources that can be colored. There is no limit to the number of colors in the scene. You can use this model with 8-, 16-, 24-, and 32-bit displays. If the display depth is less than 24 bits, the limited color resolution can produce banding artifacts; you can avoid these artifacts by using optional dithering.

The monochromatic model also supports multiple light sources, but their color content is ignored. Each source is set to a gray intensity. RGB colors at a vertex are interpreted as brightness levels, which (in Gouraud shading) are interpolated across a face between vertices with different brightnesses. The number of differently colored objects in the scene is limited; after all the system's free palette entries are used up, the system's internal palette manager finds colors that already exist in the palette and that most closely match the intended colors. Like the RGB model, you can use this model with 8-, 16-, 24-, and 32-bit displays. (The monochromatic model supports only 8-bit textures, however.) The advantage of the monochromatic model over the RGB model is simply performance.

It is not possible to change the color model of a Direct3D device. Your application should use the **IDirect3D::EnumDevices** or **IDirect3D::FindDevice** method to identify a driver that supports the required color model, then specify this driver in one of the device-creation methods.

Palettes are supported for textures, off-screen surfaces, and overlay surfaces, none of which is required to have the same palette as the primary surface. If a device supports a 4-bit indexed palette (16 colors) and you have 8-bit indexed art (256 colors), Retained Mode will render the art as 4-bit by taking the first 16 entries from your palette and remapping to those. Therefore, you should put your 16 preferred colors at the front of the palette if possible.

Window Management

For correct operation, applications must inform Direct3D when the WM_MOVE, WM_PAINT, and WM_ACTIVATE messages are received from the operating system by using the **IDirect3DRMWinDevice::HandlePaint** and **IDirect3DRMWinDevice::HandleActivate** methods.

For related information, see *IDirect3DRMWinDevice*.

IDirect3DRMFace and IDirect3DRMFaceArray Interfaces

A face represents a single polygon in a mesh. An application can set the color, texture, and material of the face by using the **IDirect3DRMFace::SetColor**, **IDirect3DRMFace::SetColorRGB**, **IDirect3DRMFace::SetTexture**, and **IDirect3DRMFace::SetMaterial** methods.

Faces are constructed from vertices by using the **IDirect3DRMFace::AddVertex** and **IDirect3DRMFace::AddVertexAndNormalIndexed** methods. An application can read the vertices of a face by using the **IDirect3DRMFace::GetVertices** and **IDirect3DRMFace::GetVertex** methods.

For related information, see *IDirect3DRMFace*.

IDirect3DRMFrame, IDirect3DRMFrame2, and IDirect3DRMFrameArray Interfaces

The term *frame* is derived from an object's physical frame of reference. The frame's role in Retained Mode is similar to a window's role in a windowing system. Objects can be placed in a scene by stating their spatial relationship to a relevant reference frame; they are not simply placed in world space. A frame is used to position objects in a scene, and visuals take their positions and orientation from frames.

A *scene* in Retained Mode is defined by a frame that has no parent frame; that is, a frame at the top of the hierarchy of frames. This frame is also sometimes called a *root frame* or *master frame*. The scene defines the frame of reference for all of the other objects. You can create a scene by calling the **IDirect3DRM::CreateFrame** method and specifying NULL for the first parameter.

The **IDirect3DRMFrame2** interface is an extension of the **IDirect3DRMFrame** interface. **IDirect3DRMFrame2** has methods that enable using materials, bounding boxes, and axes with frames. **IDirect3DRMFrame2** also supports ray picking.

By using the **IDirect3DRMFrame2::SetAxes** method and using the right-handed projection types in the **D3DRMPROJECTIONTYPE** structure with the **IDirect3DRMViewport::SetProjection** method, you can enable right-handed projection.

For related information, see *IDirect3DRMFrame* and *IDirect3DRMFrame2*.

This section describes frames and how your application can use them.

- *Hierarchies*

- *Transformations*
- *Motion*
- *Callback Functions*

Hierarchies

The frames in a scene are arranged in a tree structure. Frames can have a parent frame and child frames. Remember, a frame that has no parent frame defines a scene and is called a *root frame*.

Child frames have positions and orientations relative to their parent frames. If the parent frame moves, the child frames also move.

An application can set the position and orientation of a frame relative to any other frame in the scene, including the root frame if it needs to set an absolute position. You can also remove frames from one parent frame and add them to another at any time by using the **IDirect3DRMFrame::AddChild** method. To remove a child frame entirely, use the **IDirect3DRMFrame::DeleteChild** method. To retrieve a frame's child and parent frames, use the **IDirect3DRMFrame::GetChildren** and **IDirect3DRMFrame::GetParent** methods.

You can add frames as visuals to other frames, allowing you to use a given hierarchy many times throughout a scene. The new hierarchies are referred to as *instances*. Be careful to avoid instancing a parent frame into its children, because that will degrade performance. Retained Mode does no run-time checking for cyclic hierarchies. You cannot create a cyclic hierarchy by using the methods of the *IDirect3DRMFrame* interface; instead, this is possible only when you add a frame as a visual.

Transformations

You can think of the position and orientation of a frame relative to its parent frame as a linear transformation. This transformation takes vectors defined relative to the child frame and changes them to equivalent vectors defined relative to the parent.

Transformations can be represented by 4×4 matrices, and coordinates can be represented by four-element row vectors, [*x, y, z,*1].

If v_{child} is a coordinate in the child frame, then v_{parent}, the equivalent coordinate in the parent frame, is defined as:

$$v_{parent} = v_{child} T_{child}$$

T_{child} is the child frame's transformation matrix.

The transformations of all the parent frames above a child frame up to the root frame are concatenated with the transformation of that child to produce a world transformation. This world transformation is then applied to the visuals on the child frame before rendering. Coordinates relative to the child frame are sometimes called *model coordinates*. After the world transformation is applied, coordinates are called *world coordinates*.

The transformation of a frame can be modified directly by using the **IDirect3DRMFrame::AddTransform**, **IDirect3DRMFrame::AddScale**, **IDirect3DRMFrame::AddRotation**, and **IDirect3DRMFrame::AddTranslation** methods. Each of these methods specifies a member of the **D3DRMCOMBINETYPE** enumerated type, which specifies how the matrix supplied by the application should be combined with the current frame's matrix.

The **IDirect3DRMFrame::GetRotation** and **IDirect3DRMFrame::GetTransform** methods allow you to retrieve a frame's rotation axis and transformation matrix. To change the rotation of a frame, use the **IDirect3DRMFrame::SetRotation** method.

Use the **IDirect3DRMFrame::Transform** and **IDirect3DRMFrame::InverseTransform** methods to change between world coordinates and model coordinates.

You can find a more general discussion of transformations in the section devoted to viewports, *Transformations*. For an overview of the mathematics of transformations, see *3D Transformations*.

Motion

Every frame has an intrinsic rotation and velocity. Frames that are neither rotating nor translating simply have zero values for these attributes. These attributes are used before each scene is rendered to move objects in the scene, and they can also be used to create simple animations.

Callback Functions

Frames support a callback function that you can use to support more complex animations. The application registers a function that the frame calls before the motion attributes are applied. Where there are multiple frames in a hierarchy, each with associated callback functions, the parent frames are called before the child frames. For a given hierarchy, rendering does not take place until all of the required callback functions have been invoked.

To add this callback function, use the **IDirect3DRMFrame::AddMoveCallback** method; to remove it, use the **IDirect3DRMFrame::DeleteMoveCallback** method.

You can use these callback functions to provide new positions and orientations from a preprogrammed animation sequence or to implement dynamic motion in which the activities of visuals depend upon the positions of other objects in the scene.

IDirect3DRMInterpolator Interface

Interpolators provide a way of storing actions and applying them to objects with automatic calculation of in-between values. For example, you can set a scene's background color to red at time zero and green at time ten, and the interpolator will automatically tint successive scenes to blend from red to green. With an interpolator, you can blend colors, move objects smoothly between positions, morph meshes, and perform many other transformations.

In the Direct3D Retained-Mode implementation, interpolators are a generalization of the **IDirect3DRMAnimation** interface that increases the kinds of object parameters you can animate. While the **IDirect3DRMAnimation** interface allows animation of an object's position, size and orientation, the **IDirect3DRMInterpolator** interface further enables animation of colors, meshes, textures, and materials.

Interpolator Keys

The actions stored by the interpolator are called keys. A key is a stored procedure call and has an index associated with it. The interpolator automatically calculates between the key values.

Keys are stored in the interpolator by calling one of the supported interface methods that can be interpolated. The method and the parameter values passed to it make up the key. *Methods Supported by the Interpolator* supplies a list of supported methods.

Every key stored inside an interpolator has an index value. When the key is recorded, it is stamped with the current interpolator index value. The key's index value never changes once this value is set.

Interpolator Types

Objects can be attached to interpolators of an associated type; for example, a Mesh can be attached to a MeshInterpolator. The interpolator types are:

- FrameInterpolator
- LightInterpolator
- MaterialInterpolator
- MeshInterpolator
- TextureInterpolator

- ViewportInterpolator

Other interpolators can also be attached to an interpolator. When you change the index of an interpolator, it sets the indices of any attached interpolators to the same value.

Note that for MeshInterpolators, you add a **SetVertices** key to a MeshInterpolator object by calling **SetVertices** on the MeshInterpolator object's **IDirect3DRMMesh** interface. The group index used with **SetVertices** must correspond to a valid group index in the Mesh object or objects that the interpolator is applied to.

Interpolator Example

As an example, if you want to interpolate a frame's position, you will need a FrameInterpolator object with two interfaces, **IDirect3DRMInterpolator** and **IDirect3DRMFrame**.

```
pd3drm->CreateObject(CLSID_CDirect3DRMFrameInterpolator, 0,
IID_IDirect3DRMInterpolator, &pInterp);
pInterp->QueryInterface(IID_IDirect3DRMFrame, &pFrameInterp);
```

To add a position key to the interpolator, set the interpolator's internal index through the **IDirect3DRMInterpolator** interface, and record the position by calling the **IDirect3DRMFrame::SetPosition** method on the **IDirect3DRMFrame** interface. This method is applied to the interpolator rather than to a real frame. The function call and its parameters are stored in the interpolator as a new key with the current index.

```
pInterp->SetIndex(keytime);
pFrameInterp->SetPosition(NULL, keypos.x, keypos.y, keypos.z);
```

You can add more keys by repeating the sequence of setting the index with **SetIndex** followed by one or more object methods. To play actions back through a real frame, attach the frame to the interpolator.

```
pInterp->AttachObject(pRealFrame);
```

Now call **Interpolate** to set the position of the *pRealFrame* parameter using the interpolated position.

```
pInterp->Interpolate(time, NULL, D3DRMINTERPOLATIONSPLINE |
D3DRMINTERPOLATION_OPEN);
```

The interpolator will call the attached frame's **SetPosition** method, passing it a position it has calculated by interpolating (in this case, using a B-spline) between the nearest **SetPosition** keys.

Alternatively, you can use the immediate form of **Interpolate** and pass the object as the second parameter. This overrides any attached objects.

```
pInterp->Interpolate(time, pRealFrame, D3DRMINTERPOLATIONSPLINE |
D3DRMINTERPOLATION_OPEN);
```

You can use the same interpolator to store other keys such as orientation, scale, velocity, and color keys. Each property exists on a parallel timeline, and calling **Interpolate** assigns the interpolated value for each property to the attached frames.

It is possible to interpolate more than one method. For example, you can store **SetGroupColor** and **SetVertices** keys in the same interpolator. It is not possible to interpolate between keys of different methods, so they are stored in parallel execution threads called Key Chains. Also, if you specify two keys from different groups, such as **SetGroupColor**(0, black) and **SetGroupColor**(2, white), it does not make sense for the interpolator to generate an in-between action of **SetGroupColor**(1, gray) because the keys apply to different groups. In this case, the keys are also stored in separate chains.

Methods Supported by the Interpolator

Viewport

SetFront(value)

SetBack(value)

SetField(value)

SetPlane(left, right, bottom, top)

Frame and Frame2

SetPosition(reference*, x, y, z)

SetRotation(reference*, x, y, z, theta)

SetVelocity(reference*, x, y, z, withRotation*)

SetOrientation(reference*, dx, dy, dz, ux, uy, uz)

SetColor(color)

SetColorRGB(red, green, blue)

SetSceneBackground(color)

SetSceneBackgroundRGB(red, green, blue)

SetSceneFogColor(color)

SetSceneFogParams(start, end, density)

SetQuaternion(reference*, quat)

Mesh

Translate(x, y, z)

SetVertices(group*, index*, count*, vertices)

SetGroupColor(group*, color)

SetGroupColorRGB(group*, red, green, blue)

Light

SetColor(color)

SetColorRGB(red, green, blue)

SetRange(value)

SetUmbra(value)

SetPenumbra(value)

SetConstantAttenuation(value)

SetLinearAttenuation(value)

SetQuadraticAttenuation(value)

Texture and Texture2

SetDecalSize(width, height)

SetDecalOrigin(x, y)

SetDecalTransparentColor(color)

Material

SetPower(value)

SetSpecular(red, green, blue)

SetEmissive(red, green, blue)

*—Indicates keys with different values for this parameter are inserted in separate chains

An attempt to set a key of any unsupported method will result in a non-fatal D3DRMERR_BADOBJECT error.

Interpolator Index Span

The interpolator covers a span of index values. This index span is dictated by the following rules:

- The start of the span is the minimum of all key index values and the current index.

- The end of the span is the maximum of all key index values and the current index.

Interpolation Options

Interpolation can be performed with one or more of the following options:

D3DRMINTERPOLATION_CLOSED

D3DRMINTERPOLATION_LINEAR

D3DRMINTERPOLATION_NEAREST

D3DRMINTERPOLATION_OPEN

D3DRMINTERPOLATION_SLERPNORMALS

D3DRMINTERPOLATION_SPLINE

D3DRMINTERPOLATION_VERTEXCOLOR

If the interpolator is executed CLOSED, the interpolation is cyclic. The keys effectively repeat infinitely with a period equal to the index span. For compatibility with animations, any key with an index equal to the end of the span is ignored.

If the interpolation is OPEN, the first and last keys of each key chain fix the interpolated values outside of the index span.

The NEAREST, LINEAR, and SPLINE options specify how in-betweening is performed on each key chain. If NEAREST is specified the nearest key value is used. LINEAR performs linear interpolation between the 2 nearest keys. SPLINE uses a B-spline blending function on the 4 nearest keys.

The following two options affect only the interpolation of **IDirect3DRMMesh::SetVertices**:

- VERTEXCOLOR specifies that vertex colors should be interpolated.
- SLERPNORMALS specifies that vertex normals should be spherically interpolated (not currently implemented).

IDirect3DRMLight and IDirect3DRMLightArray Interfaces

Lighting effects are employed to increase the visual fidelity of a scene. The system colors each object based on the object's orientation to the light sources in the scene. The contribution of each light source is combined to determine the color of the object during rendering. All lights have color and intensity that can be varied independently.

An application can attach lights to a frame to represent a light source in a scene. When a light is attached to a frame, it illuminates visual objects in the scene. The

frame provides both position and orientation for the light. In other words, the light originates from the origin of the frame it is attached to. An application can move and redirect a light source simply by moving and reorienting the frame the light source is attached to.

Each viewport owns one or more lights. No light can be owned by more than one viewport.

Retained Mode currently provides five types of light sources: ambient, directional, parallel point, point, and spotlight.

For a reference to the methods of the **IDirect3DRMLight** interface, see *IDirect3DRMLight*.

This section describes lighting effects available in Direct3D and how your application can use them.

- *Ambient*
- *Directional*
- *Parallel Point*
- *Point*
- *Spotlight*

Ambient

An *ambient* light source illuminates everything in the scene, regardless of the orientation, position, and surface characteristics of the objects in the scene. Because ambient light illuminates a scene with equal strength everywhere, the position and orientation of the frame it is attached to are inconsequential. Multiple ambient light sources are combined within a scene.

Directional

A *directional* light source has orientation but no position. The light is attached to a frame but appears to illuminate all objects with equal intensity, as if it were at an infinite distance from the objects. The directional source is commonly used to simulate distant light sources, such as the sun. It is the best choice of light to use for maximum rendering speed.

Parallel Point

A *parallel point* light source illuminates objects with parallel light, but the orientation of the light is taken from the position of the parallel point light source. That is, like a directional light source, a parallel point light source has orientation, but it also has position. For example, two meshes on either side of a parallel point light source are lit on the side that faces the position of the source. The parallel

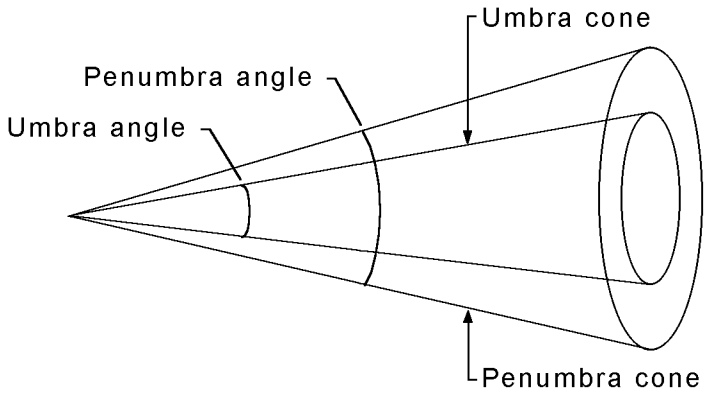
point light source offers similar rendering-speed performance to the directional light source.

Point

A *point* light source radiates light equally in all directions from its origin. It requires the calculation of a new lighting vector for every facet or normal it illuminates, and for this reason it is computationally more expensive than a parallel point light source. It does, however, produce a more faithful lighting effect and should be chosen where visual fidelity is the deciding concern.

Spotlight

A *spotlight* emits a cone of light. Only objects within the cone are illuminated. The cone produces light of two degrees of intensity, with a central brightly lit section (the *umbra*) that acts as a point source, and a surrounding dimly lit section (the *penumbra*) that merges with the surrounding deep shadow. The angles of these two sections can be individually specified by using the **IDirect3DRMLight::GetPenumbra**, **IDirect3DRMLight::GetUmbra**, **IDirect3DRMLight::SetPenumbra**, and **IDirect3DRMLight::SetUmbra** methods.



IDirect3DRMMaterial Interface

A material defines how a surface reflects light. A material has two components: an *emissive property* (whether it emits light) and a *specular property*, whose brightness is determined by a *power* setting. The value of the power determines the sharpness of the reflected highlights, with a value of 5 giving a metallic appearance and higher values giving a more plastic appearance.

An application can control the emission of a material by using the **IDirect3DRMMaterial::GetEmissive** and **IDirect3DRMMaterial::SetEmissive** methods, the specular component by using

the **IDirect3DRMMaterial::GetSpecular** and **IDirect3DRMMaterial::SetSpecular** methods, and the power by using the **IDirect3DRMMaterial::GetPower** and **IDirect3DRMMaterial::SetPower** methods.

For a reference to the methods of the **IDirect3DRMMaterial** interface, see *IDirect3DRMMaterial*.

IDirect3DRMMesh, IDirect3DRMMeshBuilder, and IDirect3DRMMeshBuilder2 Interfaces

A mesh is a visual object that is made up of a set of polygonal faces. A mesh defines a set of vertices and a set of faces (the faces are defined in terms of the vertices and normals of the mesh). Changing a vertex or normal that is used by several faces changes the appearance of all faces sharing it.

The vertices of a mesh define the positions of faces in the mesh, and they can also be used to define 2-D coordinates within a texture map.

You can manipulate meshes in Retained Mode by using three COM interfaces: *IDirect3DRMMesh*, *IDirect3DRMMeshBuilder*, and *IDirect3DRMMeshBuilder2*. **IDirect3DRMMesh** is very fast, and you should use it when a mesh is subject to frequent changes, such as when morphing. **IDirect3DRMMeshBuilder** is built on top of the **IDirect3DRMMesh** interface. Although the **IDirect3DRMMeshBuilder** interface is a convenient way to perform operations on individual faces and vertices, the system must convert a Direct3DRMMeshBuilder object into a Direct3DRMMesh object before rendering it. For meshes that do not change or that change infrequently, this conversion has a negligible impact on performance.

IDirect3DRMMeshBuilder2 has all the functionality of **IDirect3DRMMeshBuilder** plus one enhanced and one added method. **IDirect3DRMMeshBuilder2::GenerateNormals2** gives you more control over how normals are generated. **IDirect3DRMMeshBuilder2::GetFace** allows you to access a single face in a mesh.

If an application needs to assign the same characteristics (such as material or texture) to several vertices or faces, it can use the **IDirect3DRMMesh** interface to combine them in a group. If the application needs to share vertices between two different groups (for example, if neighboring faces in a mesh are different colors), the vertices must be duplicated in both groups. The **IDirect3DRMMesh::AddGroup** method assigns a group identifier to a collection of faces. This identifier is used to refer to the group in subsequent calls.

The **IDirect3DRMMeshBuilder**, **IDirect3DRMMeshBuilder2**, and **IDirect3DRMMesh** interfaces allow an application to create faces with more than three sides. They also automatically split a mesh into multiple buffers if, for example, the hardware the application is rendering to has a limit of 64K and a mesh is larger than that size. These features set the Direct3DRMMesh and Direct3DRMMeshBuilder API apart from the Direct3D API.

You can add vertices and faces individually to a mesh by using the **IDirect3DRMMeshBuilder::AddVertex**, **IDirect3DRMMeshBuilder::AddFace**, and **IDirect3DRMMeshBuilder::AddFaces** methods or the equivalent **IDirect3DRMMeshBuilder2** methods. You can retrieve an individual face with the **IDirect3DRMMeshBuilder2::GetFace**.

You can define individual color, texture, and material properties for each face in the mesh, or for all faces in the mesh at once, by using the **IDirect3DRMMesh::SetGroupColor**, **IDirect3DRMMesh::SetGroupColorRGB**, **IDirect3DRMMesh::SetGroupTexture**, and **IDirect3DRMMesh::SetGroupMaterial** methods.

For a mesh to be rendered, you must first add it to a frame by using the **IDirect3DRMFrame::AddVisual** method. You can add a single mesh to multiple frames to create multiple instances of that mesh.

Your application can use flat, Gouraud, and Phong shade modes, as specified by a call to the **IDirect3DRMMesh::SetGroupQuality** method. (Phong shading is not yet available, however.) This method uses values from the **D3DRMRENDERQUALITY** enumerated type. For more information about shade modes, see *Polygons*.

You can set normals (which should be unit vectors), or normals can be calculated by averaging the face normals of the surrounding faces by using the **IDirect3DRMMeshBuilder::GenerateNormals** method.

Direct3DRMObject

Direct3DRMObject is the common superclass of all objects in the system. A Direct3DRMObject object has characteristics common to all objects.

Each Direct3DRMObject object is instantiated as a COM object. In addition to the methods of the *IUnknown* interface, each object has a standard set of methods that are generic to all.

To create an object, the application must first have instantiated a Direct3D Retained-Mode object by calling the **Direct3DRMCreate** function. The application then calls the method of the object's interface that creates an object, and it specifies parameters specific to the object. For example, to create a

Direct3DRMAnimation object, the application would call the **IDirect3DRM::CreateAnimation** method. The creation method then creates a new object, initializes some of the object's attributes from data passed in the parameters (leaving all others with their default values), and returns the object. Applications can then specify the interface for this object to modify and use the object.

Any object can store 32 bits of application-specific data. This data is not interpreted or altered by Retained Mode. The application can read this data by using the **IDirect3DRMObject::GetAppData** method, and it can write to it by using the **IDirect3DRMObject::SetAppData** method. Finding this data is simpler if the application keeps a structure for each Direct3DRMFrame object. For example, if calling the **IDirect3DRMFrame::GetParent** method retrieves a Direct3DRMFrame object, the application can easily retrieve the data by using a pointer to its private structure, possibly avoiding a time-consuming search.

You might also want to assign a name to an object to help you organize an application or as part of your application's user interface. You can use the **IDirect3DRMObject::SetName** and **IDirect3DRMObject::GetName** methods to set and retrieve object names.

Another example of possible uses for application-specific data is when an application needs to group the faces within a mesh into subsets (for example, for front and back faces). You could use the application data in the face to note in which of these groups a face should be included.

An application can specify a function to call when an object is destroyed, such as when the application needs to deallocate memory associated with the object. To do this, use the **IDirect3DRMObject::AddDestroyCallback** method. To remove a function previously registered with this method, use the **IDirect3DRMObject::DeleteDestroyCallback** method.

The callback function is called only when the object is destroyed—that is, when the object's reference count has reached 0 and the system is about to deallocate the memory for the object. If an application kept additional data about an object (so that its dynamics could be implemented, for example), the application could use this callback function as a way to notify itself that it can dispose of the data.

For related information, see **IDirect3DRMObject** and **IDirect3DRMObjectArray**.

IDirect3DRMPickedArray and IDirect3DRMPicked2Array Interfaces

Picking is the process of searching for visuals in a scene, given a 2-D coordinate in a viewport or a vector in a frame.

You can use the **IDirect3DRMViewport::Pick** method to retrieve an **IDirect3DRMPickedArray** interface, and then call the **IDirect3DRMPickedArray::GetPick** method to retrieve an **IDirect3DRMFrameArray** interface and a visual object. The array of frames is the path through the hierarchy leading to the visual object; that is, a hierarchical list of the visual object's parent frames, with the topmost parent in the hierarchy first in the array.

You can use the **IDirect3DRMFrame2::RayPick** method to retrieve an **IDirect3DRMPicked2Array** interface, and then call the **IDirect3DRMPicked2Array::GetPick** method to retrieve an **IDirect3DRMFrameArray** interface, a visual object, and information about the object intersected by the ray, including the face and group identifiers, pick position, and horizontal and vertical texture coordinates for the vertex, vertex normal, and color of the object. The array of frames is the path through the hierarchy leading to the visual object.

IDirect3DRMProgressiveMesh Interface

A mesh is a visual object that is made up of a set of polygonal faces. A mesh defines a set of vertices and a set of faces.

A progressive mesh is a mesh that is stored as a base mesh (a coarse version) and a set of records that are used to increasingly refine the mesh. This allows you to set the level of detail rendered for a mesh and also allows progressive download from remote sources.

Using the methods of the **IDirect3DRMProgressiveMesh** interface, you can set the number of vertices or faces to render and thereby control the render detail. You can also specify a minimum level of detail required for rendering. Normally, a progressive mesh is rendered once the base mesh is available, but with the **IDirect3DRMProgressiveMesh::SetMinRenderDetail** method you can specify that a greater level of detail is necessary before rendering. You can also build a **Direct3DRMMesh** object from a particular state of the progressive mesh using the **IDirect3DRMProgressiveMesh::CreateMesh** method.

You can load a progressive mesh from a file, resource, memory, or URL. Loading can be done synchronously or asynchronously. You can check the status of a download with the **IDirect3DRMProgressiveMesh::GetLoadStatus** method, and terminate a download with the **IDirect3DRMProgressiveMesh::Abort** method. If loading is asynchronous, it is up to the application to use events through the **IDirect3DRMProgressiveMesh::RegisterEvents** and **IDirect3DRMProgressiveMesh::GetLoadStatus** methods to find out how the load is progressing.

IDirect3DRMShadow Interface

Applications can produce an initialized and usable shadow simply by calling the **IDirect3DRM::CreateShadow** method. The **IDirect3DRMShadow** interface exists so that applications which create a shadow by using the **IDirect3DRM::CreateObject** method can initialize the shadow by calling the **IDirect3DRMShadow::Init** method.

IDirect3DRMTexture and IDirect3DRMTexture2 Interfaces

A texture is a rectangular array of colored pixels. (The rectangle does not necessarily have to be square, although the system deals most efficiently with square textures.) You can use textures for texture-mapping faces, in which case their dimensions must be powers of two.

Your application can use the **IDirect3DRM::CreateTexture** method to create a texture from a **D3DRMIMAGE** structure, or the **IDirect3DRM::CreateTextureFromSurface** method to create a texture from a DirectDraw surface. The **IDirect3DRM::LoadTexture** method allows your application to load a texture from a file; the texture should be in Windows bitmap (.bmp) or Portable Pixmap (.ppm) format. To avoid unnecessary delays when creating textures, hold onto textures you want to use again, instead of creating them each time they're needed. For optimal performance, use a texture surface format that is supported by the device you are using. This will avoid a costly format conversion when the texture is created and any time it changes.

The texture coordinates of each face define the region in the texture that is mapped onto that particular face. Your application can use a wrap to calculate texture coordinates. For more information, see *IDirect3DRMWrap Interface*.

The **IDirect3DRMTexture2** interface is an extension of the **IDirect3DRMTexture** interface. The **IDirect3DRMTexture2::InitFromResource2** method allows resources to be loaded from DLLs and executables other than the currently executing file. In addition, **IDirect3DRMTexture2** has two new methods. **IDirect3DRMTexture2::InitFromImage** creates a texture from an image in memory. This method is equivalent to **IDirect3DRM::CreateTexture**. **IDirect3DRMTexture2::GenerateMIPMap** generates a MIP map from a source image.

Textures are loaded from BMP and DIB (device-independent bitmap) files right-side up in **IDirect3DRMTexture2::InitFromFile** and **IDirect3DRMTexture2::InitFromResource2**, unlike **IDirect3DRMTexture::InitFromFile** and **IDirect3DRMTexture::InitFromResource** where they are loaded inverted.

For a reference to the methods of these interfaces, see *IDirect3DRMTexture* and *IDirect3DRMTexture2*.

This section describes the types of textures supported by Direct3D and how your application can use them.

- *Decals*
- *Texture Colors*
- *Mipmaps*
- *Texture Filtering*
- *Texture Transparency*
- *Texture Format Selection Rules*

Decals

Textures can also be rendered directly, as visuals. Textures used this way are sometimes known as *decals*, a term adopted by Retained Mode. A decal is rendered into a viewport-aligned rectangle. The rectangle can optionally be scaled by the depth component of the decal's position. The size of the decal is taken from a rectangle defined relative to the containing frame by using the **IDirect3DRMTexture::SetDecalSize** method. (An application can retrieve the size of the decal by using the **IDirect3DRMTexture::GetDecalSize** method.) The decal is then transformed and perspective projection is applied.

Decals have origins that your application can set and retrieve by using the **IDirect3DRMTexture::SetDecalOrigin** and **IDirect3DRMTexture::GetDecalOrigin** methods. The origin is an offset from the top-left corner of the decal. The default origin is [0, 0]. The decal's origin is aligned with its frame's position when rendering.

Texture Colors

You can set and retrieve the number of colors that are used to render a texture by using the **IDirect3DRMTexture::SetColors** and **IDirect3DRMTexture::GetColors** methods.

If your application uses the RGB color model, you can use 8-bit, 24-bit, and 32-bit textures. If you use the monochromatic (or ramp) color model, however, you can use only 8-bit textures.

Several shades of each color are used when lighting a scene. An application can set and retrieve the number of shades used for each color by calling the **IDirect3DRMTexture::SetShades** and **IDirect3DRMTexture::GetShades** methods.

A Direct3DRMTexture object uses a **D3DRMIMAGE** structure to define the bitmap that the texture will be rendered from. If the application provides the

D3DRMIMAGE structure, the texture can easily be animated or altered during rendering.

Mipmaps

A mipmap is a sequence of textures, each of which is a progressively lower resolution, prefiltered representation of the same image. Mipmapping is a computationally low-cost way of improving the quality of rendered textures. Each prefiltered image, or level, in the mipmap is a power of two smaller than the previous level. You can specify mipmaps when filtering textures by calling the **IDirect3DRMDevice::SetTextureQuality** method.

For more information about mipmaps, see *Mipmaps*.

Texture Filtering

After a texture has been mapped to a surface, the texture elements (*texels*) of the texture rarely correspond to individual pixels in the final image. A pixel in the final image can correspond to a large collection of texels or to a small piece of a single texel. You can use texture filtering to specify how to interpolate texel values to pixels.

You can use the **IDirect3DRMDevice::SetTextureQuality** method and the **D3DRMTEXTUREQUALITY** enumerated type to specify the texture filtering mode for your application.

Texture Transparency

You can use the **IDirect3DRMTexture::SetDecalTransparency** method to produce transparent textures. Another method for achieving transparency is to use DirectDraw's support for *color keys*. Color keys are colors or ranges of colors that can be part of either the source or destination of a blit or overlay operation. You can specify that these colors should always be overwritten or never be overwritten.

For more information about DirectDraw's support for color keys, see *Color Keying*.

Texture Format Selection Rules

When you use a device-independent source image to create a device-dependent texture surface for rendering, the rules are (in order of precedence):

1. Preserve RGB/palettized nature

2. Preserve alpha channel

3. Preserve bit depth or palette size

4. Preserve RBGA masks

5. Prefer 8-bit palettized or 16-bit RGB

For more information about texture pixel formats, see *Texture Map Formats*.

For related information, see **IDirect3DRMTexture** and **IDirect3DRMTexture2**.

IDirect3DRMUserVisual Interface

User-visual objects are application-defined data that an application can add to a scene and then render, typically by using a customized rendering module. For example, an application could add sound as a user-visual object in a scene, and then render the sound during playback.

You can use the **IDirect3DRM::CreateUserVisual** method to create a user-visual object and the **IDirect3DRMUserVisual::Init** method to initialize the object.

IDirect3DRMViewport and IDirect3DRMViewportArray Interface

The viewport defines how the 3-D scene is rendered into a 2-D window. The viewport defines a rectangular area on a device that objects will be rendered into.

For a reference to the methods of this interface, see *IDirect3DRMViewport*.

This section describes the viewport, its components, and techniques for their use.

- *Camera*
- *Viewing Frustum*
- *Transformations*
- *Picking*

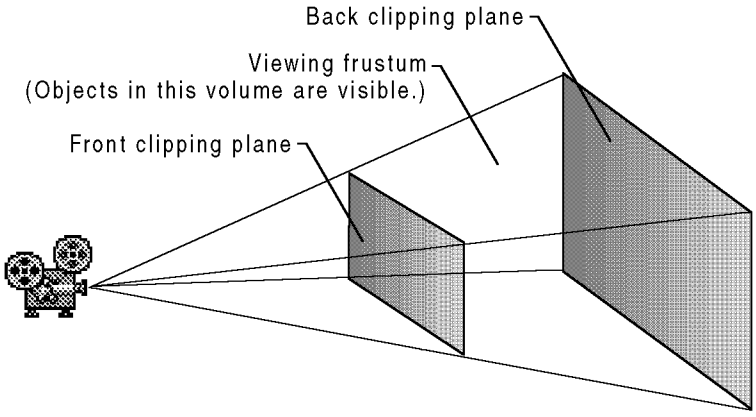
Camera

The viewport uses a Direct3DRMFrame object as a *camera*. The camera frame defines which scene is rendered and the viewing position and direction. The viewport renders only what is visible along the positive z-axis of the camera frame, with the up direction being in the direction of the positive y-axis.

An application can call the **IDirect3DRMViewport::SetCamera** method to set a camera for a given viewport. This method sets a viewport's position, direction, and orientation to that of the given camera frame. To retrieve the current camera settings, call the **IDirect3DRMViewport::GetCamera** method.

Viewing Frustum

The *viewing frustum* is a 3-D volume in a scene positioned relative to the viewport's camera. For perspective viewing, the camera is positioned at the tip of an imaginary pyramid. This pyramid is intersected by two clipping planes, the front clipping plane and the back clipping plane. The volume in the pyramid between the front and back clipping planes is the viewing frustum. Only objects in the viewing frustum are visible.



The z-axis of the camera runs from the tip of the pyramid to the center of the back clipping plane. Your application can set and retrieve the positions of the front and back clipping planes by using the **IDirect3DRMViewport::SetFront**, **IDirect3DRMViewport::SetBack**, **IDirect3DRMViewport::GetFront**, and **IDirect3DRMViewport::GetBack** methods.

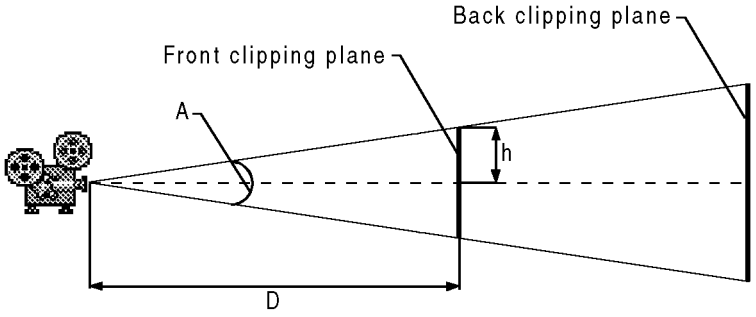
The dimensions of the viewport on the front clipping plane determine the apparent focal length of the camera's lens. (You can also think of this as a way to set the magnification of objects in the frustum.) To set and retrieve proportional dimensions for the viewport on the front clipping plane, use the **IDirect3DRMViewport::SetField** and **IDirect3DRMViewport::GetField** methods. To set and retrieve arbitrary dimensions for the viewport on the front clipping plane, use the **IDirect3DRMViewport::SetPlane** and **IDirect3DRMViewport::GetPlane** methods.

You can use the following equation to determine the relationship between the height (or width) of the front clipping plane and the viewing angle:

$$A = 2\tan^{-1}\frac{h}{D}$$

In this formula, the viewing angle is A, the front clipping plane is a distance D from the camera, and the height or width of the front clipping plane is $2h$. If the

device is not square, and thus the clipping planes are not square, the viewing angle is calculated using half the height or half the width of the front clipping plane, whichever is larger. The scale factors are set to the major axis of the device so you don't get distorted objects. If this is not what you want, you need to set uniform scaling.



The viewing frustum is a pyramid only for perspective viewing. For orthographic viewing, the viewing frustum is cuboid. These viewing types (or projection types) are defined by the **D3DRMPROJECTIONTYPE** enumerated type and used by the **IDirect3DRMViewport::GetProjection** and **IDirect3DRMViewport::SetProjection** methods.

Transformations

To render objects with 3-D coordinates in a 2-D window, the object must be transformed into the camera's frame. A projection matrix is then used to give a four-element homogeneous coordinate [*x y z w*], which is used to derive a three-element coordinate [*x/w y/w z/w*], where [*x/w y/w*] is the coordinate to be used in the window and *z/w* is the depth, ranging from 0 at the front clipping plane to 1 at the back clipping plane. The projection matrix is a combination of a perspective transformation followed by a scaling and translation to scale the objects into the window.

The following values are the elements of the projection matrix. In these formulas, *h* is the half-height of the viewing frustum, *F* is the distance from the camera, and *D* is the position in z-coordinates of the front clipping plane:

$$P = \begin{bmatrix} D/hF & 0 & 0 & 0 \\ 0 & D/hF & 0 & 0 \\ 0 & 0 & F/(F-D) & 1 \\ 0 & 0 & (-F*D)/(F-D) & 0 \end{bmatrix}$$

After projection, the next step is clipping and the conversion of x and y to screen-pixel coordinates within a viewport. Use the D3DVIEWPORT data members for this. The viewport is a rectangular window on the rendering surface.

```
typedef struct _D3DVIEWPORT {
    DWORD    dwSize;
    DWORD    dwX;
    DWORD    dwY;
```

dwX and dwY specify the offset in screen pixels to the top left of the viewport on the surface.

```
    DWORD    dwWidth;
    DWORD    dwHeight;
```

dwWidth and dwHeight are the width and height of the viewport in screen pixels.

```
    D3DVALUE dvScaleX;
    D3DVALUE dvScaleY;
```

dvScaleX and dvScaleY are the scaling factors that are applied to the x and y values to yield screen coordinates. You would usually want to map the entire normalized perspective view volume onto the viewport using the following formulas:

```
  dvScaleX = dwWidth / 2
  dvScaleY = dwHeight / 2
```

X coordinates, for example, in the range of -1 to 1, will be scaled into the range of -dwWidth / 2 to dwWidth / 2. An offset of dwWidth / 2 is then added. This scaling occurs after clipping.

If the window is not square and you would like to preserve a correct aspect ratio, use the larger of the two window dimensions for both scaling values. You will also need to clip some of the view volume.

```
    D3DVALUE dvMaxX;
    D3DVALUE dvMaxY;
    D3DVALUE dvMinZ;
    D3DVALUE dvMaxZ;
```

These fields specify the clipping planes: x = dvMaxX, x = -dvMaxX, y = dvMaxY, y = -dvMaxY, z = dvMinZ, z = dvMaxZ. To display all of the view volume, for example, you would set dvMaxX = dvMaxY = dvMaxZ = 1 and dvMinZ = 0. As noted above, if you want to preserve the correct aspect ratio on a nonsquare window, you will need to clip some of the view volume. To do so, use

the following equations. These equations also work with square viewports, so use them all the time.

```
dvMaxX = dwWidth / (2 * dvScaleX)
dvMaxY = dwHeight / (2 * dvScaleY)

} D3DVIEWPORT, *LPD3DVIEWPORT;
```

An application uses the viewport transformation to ensure that the distance by which the object is moved in world coordinates is scaled by the object's distance from the camera to account for perspective. Note that the result from **IDirect3DRMViewport::Transform** is a four-element homogeneous vector. This avoids the problems associated with coordinates being scaled by an infinite amount near the camera's position.

For information about transformations for frames, see *Transformations*. For an overview of the mathematics of transformations, see *3D Transformations*.

Picking

Picking is the process of searching for visuals in the scene given a 2-D coordinate in the viewport's window. An application can use the **IDirect3DRMViewport::Pick** method to retrieve either the closest object in the scene or a depth-sorted list of objects.

IDirect3DRMVisual and IDirect3DRMVisualArray Interfaces

Visuals are objects that can be rendered in a scene. Visuals are visible only when they are added to a frame in that scene. An application can add a visual to a frame by using the **IDirect3DRMFrame::AddVisual** method. The frame provides the visual with position and orientation for rendering.

You should use the **IDirect3DRMVisualArray** interface to work with groups of visual objects; although there is a **IDirect3DRMVisual** COM interface, it has no methods.

The most common visual types are Direct3DRMMeshBuilder and Direct3DRMTexture objects.

IDirect3DRMWrap Interface

You can use a wrap to calculate texture coordinates for a face or mesh. To create a wrap, the application must specify a type, a reference frame, an origin, a direction vector, and an up vector. The application must also specify a pair of scaling factors and an origin for the texture coordinates.

Your application calls the **IDirect3DRM::CreateWrap** function to create an **IDirect3DRMWrap** interface. This interface has two unique methods: **IDirect3DRMWrap::Apply**, which applies a wrap to the vertices of the object, and **IDirect3DRMWrap::ApplyRelative**, which transforms the vertices of a wrap as it is applied.

In the examples, the direction vector (the v vector) lies along the z-axis, and the up vector (the u vector) lies along the y-axis, with the origin at [0 0 0].

For a reference to the methods of the **IDirect3DRMWrap** interface, see *IDirect3DRMWrap*.

This section describes the wrapping flags and the four wrapping types:

- *Wrapping Flags*
- *Flat*
- *Cylindrical*
- *Spherical*
- *Chrome*

Wrapping Flags

The **D3DRMMAPPING** type includes the D3DRMMAP_WRAPU and D3DRMMAP_WRAPV flags. These flags determine how the rasterizer interprets texture coordinates. The rasterizer always interpolates the shortest distance between texture coordinates—that is, a line. The path taken by this line, and the valid values for the u- and v-coordinates, varies with the use of the wrapping flags. If either or both flags is set, the line can wrap around the texture edge in the u or v direction, as if the texture had a cylindrical or toroidal topology.

- In flat wrapping mode, in which neither of the wrapping flags is set, the plane specified by the u- and v-coordinates is an infinite tiling of the texture. In this case, values greater than 1.0 are valid for u and v. The shortest line between (0.1, 0.1) and (0.9, 0.9) passes through (0.5, 0.5).
- If either D3DRMMAP_WRAPU or D3DRMMAP_WRAPV is set, the texture is a cylinder with an infinite length and a circumference of 1.0. Texture coordinates greater than 1.0 are valid only in the dimension that is not wrapped. The shortest distance between texture coordinates varies with the wrapping flag; if D3DRMMAP_WRAPU is set, the shortest line between (0.1, 0.1) and (0.9, 0.9) passes through (0, 0.5).
- If both D3DRMMAP_WRAPU and D3DRMMAP_WRAPV are set, the texture is a torus. Because the system is closed, texture coordinates greater that 1.0 are invalid. The shortest line between (0.1, 0.1) and (0.9, 0.9) passes through (0, 0).

Although texture coordinates that are outside the valid range may be truncated to valid values, this behavior is not defined.

Typically, applications set a wrap flag for cylindrical wraps when the intersection of the texture edges does not match the edges of the face; applications do not set a wrap flag when more than half of a texture is applied to a single face.

Flat

The flat wrap conforms to the faces of an object as if the texture were a piece of rubber that was stretched over the object.

The [*u v*] coordinates are derived from a vector [*x y z*] by using the following equations:

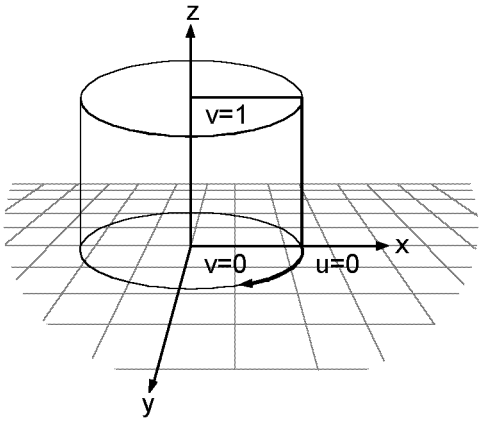
$$u = s_u x - o_u$$
$$v = s_v y - o_v$$

In these formulas, *s* is the window-scaling factor and *o* is the window origin. The application should choose a pair of scaling factors and offsets that map the ranges of *x* and *y* to the range from 0 to 1 for *u* and *v*.

Cylindrical

The cylindrical wrap treats the texture as if it were a piece of paper that is wrapped around a cylinder so that the left edge is joined to the right edge. The object is then placed in the middle of the cylinder and the texture is deformed inward onto the surface of the object.

For a cylindrical texture map, the effects of the various vectors are shown in the following illustration.



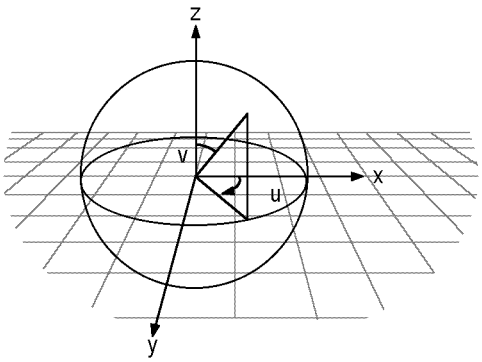
The direction vector specifies the axis of the cylinder, and the up vector specifies the point on the outside of the cylinder where *u* equals 0. To calculate the texture [*u v*] coordinates for a vector [*x y z*], the system uses the following equations:

$$u = \frac{s_u}{2\pi} \tan^{-1} \frac{x}{y} - o_y$$
$$v = s_v z - o_v$$

Typically, *u* would be left unscaled and *v* would be scaled and translated so that the range of *z* maps to the range from 0 to 1 for *v*.

Spherical

For a spherical wrap, the u-coordinate is derived from the angle that the vector [*x y* 0] makes with the x-axis (as in the cylindrical map) and the v-coordinate from the angle that the vector [*x y z*] makes with the z-axis. Note that this mapping causes distortion of the texture at the z-axis.



This translates to the following equations:

$$u = \frac{S_u}{2\pi} \tan^{-1} \frac{x}{y} - o_u$$
$$v = \frac{S_v}{\pi} \tan^{-1} \frac{z}{\sqrt{x^2 + y^2 + z^2}} - o_v$$

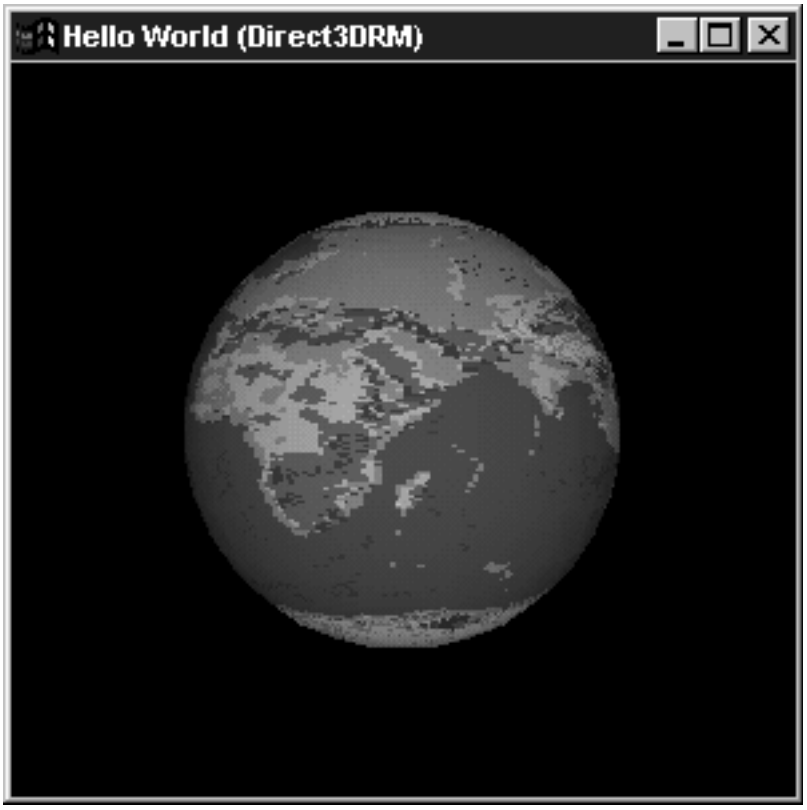
The scaling factors and texture origin will often not be needed here as the unscaled range of *u* and *v* is already 0 through 1.

Chrome

A chrome wrap allocates texture coordinates so that the texture appears to be reflected onto the objects. The chrome wrap takes the reference frame position and uses the vertex normals in the mesh to calculate reflected vectors. The texture u- and v-coordinates are then calculated from the intersection of these reflected vectors with an imaginary sphere that surrounds the mesh. This gives the effect of the mesh reflecting whatever is wrapped on the sphere.

Direct3D Retained-Mode Tutorial

To create a Windows-based Direct3D Retained-Mode application, you set up the features of two different environments: the devices, viewports, and color capabilities of the Windows environment, and the models, textures, lights, and positions of the virtual environment. This section is a tutorial that contains all the code for a simple Retained-Mode application. The following illustration is a single frame from the running animation:



The tutorial is divided into the following sections:

- *Making the Helworld Sample*
- *Definitions and Global Variables*
- *Windows Setup and Initialization*
- *Enumerating Device Drivers*
- *Setting up the 3-D Environment*
- *The Rendering Loop*
- *Creating the Scene*
- *Cleaning Up*

Making the Helworld Sample

This tutorial includes all the code for a working Direct3D Retained-Mode application. If you copy the code from this tutorial into a single .c file, you can compile and run it if:

- you have the Direct3D header files (d3d.h, d3drm.h, and so on) in your include path
- you link with the Winmm.lib, D3drm.lib, and Ddraw.lib static libraries
- your compiler can find the Sphere3.x file in the DirectX SDK's media directory
- you supply a bitmap called tutor.bmp

  Note that this bitmap must have pixel dimensions that are a power of 2; for example, a 16 by 16 pixel bitmap, a 128 by 1024 pixel bitmap, a 512 by 256 pixel bitmap, and so on.

Most of the code responsible for 3-D effects in this sample has been broken out into discrete functions so that you can modify parts of the system incrementally in your own experiments. You should look at the samples in the SDK for implementations of more complicated features of Direct3D.

The following topics describe some of the concerns that apply to the overall development of a simple Direct3D Retained-Mode application.

- *Limitations of the Sample*
- *DirectDraw's Windowed Mode*

Limitations of the Sample

This tutorial contains the Helworld.c example code, which creates a sphere, applies a texture to it, and rotates it in a window. This is the only C source file required to build this application. The only other files you need are Sphere3.x, a mesh file that ships in the Media directory of the DirectX SDK, and tutor.bmp, a bitmap you supply. (Note that you must have the Direct3D header files in your

include path and link to the Winmm.lib, D3drm.lib, and Ddraw.lib static libraries.)

This tutorial is a simplied version fo the Globe sample that is part of the DirectX SDK. The Globe sample, like all the Direct3D Retained-Mode samples in the SDK, requires the inclusion of a file named Rmmain.cpp and a number of header files. In Helworld.c, the relevant parts of Rmmain.cpp have been converted to C from C++ and integrated into the source code.

The code shown in this tutorial should not be mistaken for production code. The only possible user interactions with the program are starting it, stopping it, and minimizing the window while it is running. Most of the error checking has been removed for the sake of clarity. The purpose of this example is analogous to the purpose of the beginning program that prints "Hello, world!" on the screen: to produce output with as little confusion as possible.

DirectDraw's Windowed Mode

Nearly all Direct3D applications will use DirectDraw to display their graphics on the screen. These applications will either use DirectDraw's full-screen (exclusive) mode or its windowed mode. The code in this documentation uses windowed mode. Although the full-screen mode offers some benefits in performance and convenience, it is much easier to debug code written in windowed mode. Most developers will write their code in windowed mode and port it to full-screen mode late in the development cycle, when most of the bugs have been worked out.

Definitions and Global Variables

Below are the first lines of the Helworld.c example. Helworld.c is the only C source file required to build this application.

Notice that INITGUID must be defined prior to all other includes and defines. This is a crucial point that is sometimes missed by developers who are new to DirectX.

```
//////////////////////////////////////////////////////////////////////
//
//  Copyright (C) 1996 Microsoft Corporation. All Rights Reserved.
//
//  File: Helworld.c
//
//  Simplified Direct3D Retained-Mode example, based on
//  the "Globe" SDK sample.
//
//////////////////////////////////////////////////////////////////////

#define INITGUID           // Must precede other defines and includes
```

```
#include <windows.h>
#include <malloc.h>        // Required by memset call
#include <d3drmwin.h>

#define MAX_DRIVERS 5           // Maximum D3D drivers expected

// Global variables

LPDIRECT3DRM lpD3DRM;           // Direct3DRM object
LPDIRECTDRAWCLIPPER lpDDClipper;// DirectDrawClipper object

struct _myglobs {
    LPDIRECT3DRMDEVICE dev;     // Direct3DRM device
    LPDIRECT3DRMVIEWPORT view;  // Direct3DRM viewport through which
                                // the scene is viewed
    LPDIRECT3DRMFRAME scene;    // Master frame in which others are
                                // placed
    LPDIRECT3DRMFRAME camera;   // Frame describing the user's POV

    GUID DriverGUID[MAX_DRIVERS];     // GUIDs of available D3D drivers
    char DriverName[MAX_DRIVERS][50]; // Names of available D3D drivers
    int  NumDrivers;                  // Number of available D3D drivers
    int  CurrDriver;                  // Number of D3D driver currently
                                      // being used

    BOOL bQuit;                 // Program is about to terminate
    BOOL bInitialized;          // All D3DRM objects are initialized
    BOOL bMinimized;            // Window is minimized

    int BPP;                    // Bit depth of the current display mode

} myglobs;

// Function prototypes.

static BOOL InitApp(HINSTANCE, int);
long FAR PASCAL WindowProc(HWND, UINT, WPARAM, LPARAM);
static BOOL EnumDrivers(HWND win);
static HRESULT WINAPI enumDeviceFunc(LPGUID lpGuid,
    LPSTR lpDeviceDescription, LPSTR lpDeviceName,
    LPD3DDEVICEDESC lpHWDesc, LPD3DDEVICEDESC lpHELDesc,
    LPVOID lpContext);
static DWORD BPPToDDBD(int bpp);
static BOOL CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper,
    int driver, int width, int height);
static BOOL SetRenderState(void);
static BOOL RenderLoop(void);
static BOOL MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT view,
    LPDIRECT3DRMFRAME scene, LPDIRECT3DRMFRAME camera);
```

```
void MakeMyFrames(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME * lplpLightFrame1,
    LPDIRECT3DRMFRAME * lplpWorld_frame);
void MakeMyLights(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMLIGHT * lplpLight1, LPDIRECT3DRMLIGHT * lplpLight2);
void SetMyPositions(LPDIRECT3DRMFRAME lpScene,
    LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMFRAME lpWorld_frame);
void MakeMyMesh(LPDIRECT3DRMMESHBUILDER * lplpSphere3_builder);
void MakeMyWrap(LPDIRECT3DRMMESHBUILDER sphere3_builder,
                LPDIRECT3DRMWRAP * lpWrap);
void AddMyTexture(LPDIRECT3DRMMESHBUILDER lpSphere3_builder,
                  LPDIRECT3DRMTEXTURE * lplpTex);
static void CleanUp(void);
```

Windows Setup and Initialization

This section describes the standard setup and initialization functions in a Windows program, as implemented in the Helworld.c sample code.

- *The WinMain Function*
- *The InitApp Function*
- *The Main Window Procedure*

The WinMain Function

The WinMain function in Helworld.c has only a few lines of code that are unique to an application that uses DirectDraw and Direct3D's Retained Mode. The InitApp and CleanUp functions are standard parts of a Windows program, although in the case of Helworld, they perform some unique tasks. The most important call in WinMain, from the perspective of Direct3D, is the call to the RenderLoop function. RenderLoop is responsible for drawing each new frame of the animation. For more information about the RenderLoop function, see *The Rendering Loop*.

```
//////////////////////////////////////////////////////////////////////
//
// WinMain
// Initializes the application and enters a message loop.
// The message loop renders the scene until a quit message is received.
//
//////////////////////////////////////////////////////////////////////

int PASCAL
WinMain (HINSTANCE this_inst, HINSTANCE prev_inst, LPSTR cmdline,
    int cmdshow)
{
```

```
    MSG     msg;
    HACCEL  accel = NULL;
    int     failcount = 0;  // Number of times RenderLoop has failed

    prev_inst;
    cmdline;

    // Create the window and initialize all objects needed to begin
    // rendering.

    if (!InitApp(this_inst, cmdshow))
        return 1;

    while (!myglobs.bQuit) {

        // Monitor the message queue until there are no pressing
        // messages.

        while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
            if (!TranslateAccelerator(msg.hwnd, accel, &msg)) {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }

        // If the app is not minimized, not about to quit, and D3DRM has
        // been initialized, begin rendering.

        if (!myglobs.bMinimized && !myglobs.bQuit &&
             myglobs.bInitialized) {

            // Attempt to render a frame. If rendering fails more than
            // twice, abort execution.

            if (!RenderLoop())
                ++failcount;
            if (failcount > 2) {
                CleanUp();
                break;
            }
        }
    }
    return msg.wParam;
}
```

The InitApp Function

The initialization function in Helworld.c creates the window class and main application window, just as in most Windows applications. After that, however, it does some work that is unique to applications using DirectDraw and Direct3D.

Now InitApp retrieves the current display's bits per pixel. This information is used to help set the rendering quality for the application. For more information, see *Setting the Render State*.

InitApp then calls the locally defined EnumDrivers function to determine what Direct3D drivers are available and to choose one. For more information about enumerating drivers, see *Enumerating Device Drivers*.

Next, the code calls the **Direct3DRMCreate** function to create an *IDirect3DRM* interface. It uses this interface in the calls to **IDirect3DRM::CreateFrame** and **IDirect3DRMFrame::SetPosition** that create the scene and camera frames, and position the camera in the scene.

A DirectDrawClipper object makes it simple to manage the clipping planes that control which parts of a 3-D scene are visible. Helworld.c calls the **DirectDrawCreateClipper** function to create an interface, and then uses the **IDirectDrawClipper::SetHWnd** method to set the window handle that obtains the clipping information.

Now the InitApp function calls the locally defined CreateDevAndView function to create the Direct3D device and viewport. For more information about this function, see *Creating the Device and Viewport*.

When the entire supporting structure of a Direct3D application has been put into place, the details of the 3-D scene can be constructed. The MyScene function does this work. For more information about MyScene, see *Creating the Device and Viewport*.

Finally, just as in a standard initialization function, InitApp shows and updates the window.

```
////////////////////////////////////////////////////////////////////
//
// InitApp
// Creates window and initializes all objects necessary to begin
// rendering.
//
////////////////////////////////////////////////////////////////////

static BOOL
InitApp(HINSTANCE this_inst, int cmdshow)
{
    HWND win;
    HDC hdc;
    WNDCLASS wc;
    RECT rc;

    // Set up and register the window class.

    wc.style = CS_HREDRAW | CS_VREDRAW;
```

```
wc.lpfnWndProc = WindowProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = sizeof(DWORD);
wc.hInstance = this_inst;
wc.hIcon = LoadIcon(this_inst, "AppIcon");
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "D3DRM Example";
if (!RegisterClass(&wc))
    return FALSE;

// Initialize the global variables.

memset(&myglobs, 0, sizeof(myglobs));

// Create the window.

win =
    CreateWindow
    (   "D3DRM Example",            // Class
        "Hello World (Direct3DRM)", // Title bar
        WS_VISIBLE | WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
             WS_MINIMIZEBOX | WS_MAXIMIZEBOX,
        CW_USEDEFAULT,              // Init. x pos
        CW_USEDEFAULT,              // Init. y pos
        300,                        // Init. x size
        300,                        // Init. y size
        NULL,                       // Parent window
        NULL,                       // Menu handle
        this_inst,                  // Program handle
        NULL                        // Create parms
    );
if (!win)
    return FALSE;

// Record the current display bits-per-pixel.

hdc = GetDC(win);
myglobs.BPP = GetDeviceCaps(hdc, BITSPIXEL);
ReleaseDC(win, hdc);

// Enumerate the D3D drivers and select one.

if (!EnumDrivers(win))
    return FALSE;

// Create the D3DRM object and the D3DRM window object.

lpD3DRM = NULL;
```

```
    Direct3DRMCreate(&lpD3DRM);

    // Create the master scene frame and camera frame.

    lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, NULL, &myglobs.scene);
    lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, myglobs.scene,
        &myglobs.camera);
    myglobs.camera->lpVtbl->SetPosition(myglobs.camera, myglobs.scene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0));

    // Create a DirectDrawClipper object and associate the
    // window with it.

    DirectDrawCreateClipper(0, &lpDDClipper, NULL);
    lpDDClipper->lpVtbl->SetHWnd(lpDDClipper, 0, win);

    // Create the D3DRM device by using the selected D3D driver.

    GetClientRect(win, &rc);
    if (!CreateDevAndView(lpDDClipper, myglobs.CurrDriver, rc.right,
            rc.bottom)) {
        return FALSE;
    }

    // Create the scene to be rendered.

    if (!MyScene(myglobs.dev, myglobs.view, myglobs.scene,
            myglobs.camera))
        return FALSE;

    myglobs.bInitialized = TRUE;  // Initialization completed

    // Display the window.

    ShowWindow(win, cmdshow);
    UpdateWindow(win);

    return TRUE;
}
```

The Main Window Procedure

The Helworld.c sample has a very simple main window procedure—after all, the application allows practically no user interaction.

When the window procedure receives a WM_DESTROY message, it calls the CleanUp function, just as you would expect.

When it receives a WM_ACTIVATE message, it retrieves an *IDirect3DRMWinDevice* interface and calls the

IDirect3DRMWinDevice::HandleActivate method to ensure that the colors are correct in the active rendering window. Similarly, the function reacts to a WM_PAINT message by calling the **IDirect3DRMWinDevice::HandlePaint** method.

```
//////////////////////////////////////////////////////////////////////
//
// WindowProc
// Main window message handler.
//
//////////////////////////////////////////////////////////////////////

LONG FAR PASCAL WindowProc(HWND win, UINT msg,
    WPARAM wparam, LPARAM lparam)
{
    RECT r;
    PAINTSTRUCT ps;
    LPDIRECT3DRMWINDEVICE lpD3DRMWinDev;

    switch (msg)    {

    case WM_DESTROY:
        CleanUp();
        break;

    case WM_ACTIVATE:
        {

        // Create a Windows-specific D3DRM window device to handle this
        // message.

        LPDIRECT3DRMWINDEVICE lpD3DRMWinDev;
        if (!myglobs.dev)
            break;
        myglobs.dev->lpVtbl->QueryInterface(myglobs.dev,
            &IID_IDirect3DRMWinDevice, (void **) &lpD3DRMWinDev);
        lpD3DRMWinDev->lpVtbl->HandleActivate(lpD3DRMWinDev,
            (WORD) wparam);
        lpD3DRMWinDev->lpVtbl->Release(lpD3DRMWinDev);
        }
        break;

    case WM_PAINT:
        if (!myglobs.bInitialized || !myglobs.dev)
            return DefWindowProc(win, msg, wparam, lparam);

        // Create a Windows-specific D3DRM window device to handle this
        // message.

        if (GetUpdateRect(win, &r, FALSE)) {
```

```
            BeginPaint(win, &ps);
            myglobs.dev->lpVtbl->QueryInterface(myglobs.dev,
                &IID_IDirect3DRMWinDevice, (void **) &lpD3DRMWinDev);
            if (FAILED(lpD3DRMWinDev->lpVtbl->HandlePaint(lpD3DRMWinDev,
                ps.hdc)))
            lpD3DRMWinDev->lpVtbl->Release(lpD3DRMWinDev);
            EndPaint(win, &ps);
        }
        break;
    default:
        return DefWindowProc(win, msg, wparam, lparam);
    }
    return 0L;
}
```

Enumerating Device Drivers

Applications that use Direct3D always enumerate the available drivers and choose the one that best matches their needs. The following sections each describe one of the functions that perform this task:

- *The EnumDrivers Function*
- *The enumDeviceFunc Callback Function*
- *The BPPToDDBD Helper Function*

The EnumDrivers Function

The EnumDrivers function is called by the InitApp function just before InitApp creates the application's scene and camera.

The COM interface is really an interface to a DirectDraw object, so the first thing this enumeration function does is call the **DirectDrawCreate** function to create a DirectDrawObject. Then EnumDrivers uses the **QueryInterface** method to create an **IDirect3D** interface. Notice that the C implementation of **QueryInterface** requires that you pass the address of the interface identifier as the second parameter, not simply the constant itself (as in the C++ implementation).

The enumeration is handled by the **IDirect3D::EnumDevices** method, which depends on the locally defined enumDeviceFunc callback function. For more information about this callback function, see *The enumDeviceFunc Callback Function*.

Notice that **IDirect3D::EnumDevices** is a Direct3D method, not a Direct3DRM method; there is no enumeration method in the Retained-Mode API. This is a good example of the natural use of both Retained-Mode and Immediate-Mode methods in a single application.

```
//////////////////////////////////////////////////////////////////////
```

```
//
// EnumDrivers
// Enumerate the available D3D drivers and choose one.
//
////////////////////////////////////////////////////////////////////////

static BOOL
EnumDrivers(HWND win)
{
    LPDIRECTDRAW lpDD;
    LPDIRECT3D lpD3D;
    HRESULT rval;

    // Create a DirectDraw object and query for the Direct3D interface
    // to use to enumerate the drivers.

    DirectDrawCreate(NULL, &lpDD, NULL);
    rval = lpDD->lpVtbl->QueryInterface(lpDD, &IID_IDirect3D,
        (void**) &lpD3D);
    if (rval != DD_OK) {
        lpDD->lpVtbl->Release(lpDD);
        return FALSE;
    }

    // Enumerate the drivers, setting CurrDriver to -1 to initialize the
    // driver selection code in enumDeviceFunc.

    myglobs.CurrDriver = -1;
    lpD3D->lpVtbl->EnumDevices(lpD3D, enumDeviceFunc,
        &myglobs.CurrDriver);

    // Ensure at least one valid driver was found.

    if (myglobs.NumDrivers == 0) {
        return FALSE;
    }
    lpD3D->lpVtbl->Release(lpD3D);
    lpDD->lpVtbl->Release(lpDD);

    return TRUE;
}
```

The enumDeviceFunc Callback Function

The enumDeviceFunc callback function is of the **D3DENUMDEVICESCALLBACK** type, as defined in the D3dcaps.h header file. The system calls this function with identifiers and names for each Direct3D driver in the system, as well as the hardware and emulated capabilities of the driver.

The callback function uses the **dcmColorModel** member of the **D3DDEVICEDESC** structure to determine whether to examine the hardware or emulated driver description; if the member has been filled by the hardware description, the function consults the hardware description.

Next, the callback function determines whether the driver being enumerated can render in the current bit depth. If not, the function returns D3DENUMRET_OK to skip the rest of the process for this driver and continue the enumeration with the next driver. The callback function uses the locally defined BPPToDDBD function to compare the reported bit depth against the bits-per-pixel retrieved by the call to the **GetDeviceCaps** function in the InitApp function. (BPPToDDBD stands for bits-per-pixel to DirectDraw bit-depth.) For the code for this function, see *The BPPToDDBD Helper Function*.

If the driver being enumerated passes a few simple tests, other parts of **D3DDEVICEDESC** are examined. The callback function will choose hardware over software emulation, and RGB lighting capabilities over monochromatic lighting capabilities.

```
//////////////////////////////////////////////////////////////////////
//
// enumDeviceFunc
// Callback function that records each usable D3D driver's name
// and GUID. Chooses a driver and sets *lpContext to this driver.
//
//////////////////////////////////////////////////////////////////////

static HRESULT
WINAPI enumDeviceFunc(LPGUID lpGuid, LPSTR lpDeviceDescription,
    LPSTR lpDeviceName, LPD3DDEVICEDESC lpHWDesc,
    LPD3DDEVICEDESC lpHELDesc, LPVOID lpContext)
{
    static BOOL hardware = FALSE; // Current start driver is hardware
    static BOOL mono = FALSE;     // Current start driver is mono light
    LPD3DDEVICEDESC lpDesc;
    int *lpStartDriver = (int *)lpContext;

    // Decide which device description should be consulted.

    lpDesc = lpHWDesc->dcmColorModel ? lpHWDesc : lpHELDesc;

    // If this driver cannot render in the current display bit-depth,
    // skip it and continue with the enumeration.

    if (!(lpDesc->dwDeviceRenderBitDepth & BPPToDDBD(myglobs.BPP)))
        return D3DENUMRET_OK;

    // Record this driver's name and GUID.
```

```
    memcpy(&myglobs.DriverGUID[myglobs.NumDrivers], lpGuid,
        sizeof(GUID));
    lstrcpy(&myglobs.DriverName[myglobs.NumDrivers][0], lpDeviceName);

    // Choose hardware over software, RGB lights over mono lights.

    if (*lpStartDriver == -1) {

        // This is the first valid driver.

        *lpStartDriver = myglobs.NumDrivers;
        hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
        mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
    } else if (lpDesc == lpHWDesc && !hardware) {

        // This driver is hardware and the start driver is not.

        *lpStartDriver = myglobs.NumDrivers;
        hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
        mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
    } else if ((lpDesc == lpHWDesc && hardware ) ||
               (lpDesc == lpHELDesc && !hardware)) {
        if (lpDesc->dcmColorModel == D3DCOLOR_MONO && !mono) {

            // This driver and the start driver are the same type, and
            // this driver is mono whereas the start driver is not.

            *lpStartDriver = myglobs.NumDrivers;
            hardware = lpDesc == lpHWDesc ? TRUE : FALSE;
            mono = lpDesc->dcmColorModel & D3DCOLOR_MONO ? TRUE : FALSE;
        }
    }
    myglobs.NumDrivers++;
    if (myglobs.NumDrivers == MAX_DRIVERS)
        return (D3DENUMRET_CANCEL);
    return (D3DENUMRET_OK);
}
```

The BPPToDDBD Helper Function

The enumDeviceFunc callback function uses the BPPToDDBD helper function to convert the stored bits-per-pixel the current device supports to a form that can be compared against the bit depth for the driver being enumerated. For more information about enumDeviceFunc, see *The enumDeviceFunc Callback Function*.

```
//////////////////////////////////////////////////////////////////////
//
// BPPToDDBD
// Converts bits-per-pixel to a DirectDraw bit-depth flag.
```

```
//
//////////////////////////////////////////////////////////////////////

static DWORD
BPPToDDBD(int bpp)
{
    switch(bpp) {
        case 1:
            return DDBD_1;
        case 2:
            return DDBD_2;
        case 4:
            return DDBD_4;
        case 8:
            return DDBD_8;
        case 16:
            return DDBD_16;
        case 24:
            return DDBD_24;
        case 32:
            return DDBD_32;
        default:
            return 0;
    }
}
```

Setting up the 3-D Environment

This section describes the code in Helworld.c that establishes the 3-D environment. The following sections describe the two functions that perform this task:

- *Creating the Device and Viewport*
- *Setting the Render State*

These functions do not populate the 3-D environment with objects, frames, and lights. That is done by the MyScene function and the functions it calls. For information about filling up the 3-D environment, see *Creating the Scene*.

Creating the Device and Viewport

The Direct3D device and viewport are created as part of the application's initialization. After creating a DirectDrawClipper object, the InitApp function calls CreateDevAndView, passing as parameters the DirectDrawClipper object, the driver that was chosen, and the dimensions of the client rectangle.

The CreateDevAndView function uses the **IDirect3DRM::CreateDeviceFromClipper** method to create a Direct3DRM

device, using the driver that was selected by the enumeration process. It uses this *IDirect3DRMDevice* interface to retrieve the device's width and height, by calling **IDirect3DRMDevice::GetWidth** and **IDirect3DRMDevice::GetHeight** methods. After it has retrieved this information, it calls the **IDirect3DRM::CreateViewport** method to retrieve the *IDirect3DRMViewport* interface.

When CreateDevAndView has called the **IDirect3DRMViewport::SetBack** method to set the back clipping plane of the viewport, it calls the locally defined SetRenderState function. SetRenderState is described in the next section, *Setting the Render State*.

```
////////////////////////////////////////////////////////////////////
//
// CreateDevAndView
// Create the D3DRM device and viewport with the given D3D driver and
// with the specified size.
//
////////////////////////////////////////////////////////////////////

static BOOL
CreateDevAndView(LPDIRECTDRAWCLIPPER lpDDClipper, int driver,
    int width, int height)
{
    HRESULT rval;

    // Create the D3DRM device from this window by using the specified
    // D3D driver.

    lpD3DRM->lpVtbl->CreateDeviceFromClipper(lpD3DRM, lpDDClipper,
        &myglobs.DriverGUID[driver], width, height, &myglobs.dev);

    // Create the D3DRM viewport by using the camera frame.  Set the
    // background depth to a large number. The width and height
    // might have been slightly adjusted, so get them from the device.

    width = myglobs.dev->lpVtbl->GetWidth(myglobs.dev);
    height = myglobs.dev->lpVtbl->GetHeight(myglobs.dev);
    rval = lpD3DRM->lpVtbl->CreateViewport(lpD3DRM, myglobs.dev,
        myglobs.camera, 0, 0, width, height, &myglobs.view);
    if (rval != D3DRM_OK) {
        myglobs.dev->lpVtbl->Release(myglobs.dev);
        return FALSE;
    }
    rval = myglobs.view->lpVtbl->SetBack(myglobs.view, D3DVAL(5000.0));
    if (rval != D3DRM_OK) {
        myglobs.dev->lpVtbl->Release(myglobs.dev);
        myglobs.view->lpVtbl->Release(myglobs.view);
        return FALSE;
    }
```

```
    // Set the render quality, fill mode, lighting state,
    // and color shade info.

    if (!SetRenderState())
        return FALSE;
    return TRUE;
}
```

Setting the Render State

Direct3D is a *state machine*; applications set up the state of the lighting, rendering, and transformation modules and then pass data through them. This architecture is integral to Immediate Mode, but it is partially hidden by the Retained-Mode API. The SetRenderState function is a simple way to set the rendering state for a Retained-Mode application.

First, SetRenderState calls the **IDirect3DRMDevice::SetQuality** method, specifying that the lights are on, that the fill mode is solid, and that Gouraud shading should be used. At this point, applications that need to change the dithering mode or texture quality can call the **IDirect3DRMDevice::SetDither** or **IDirect3DRMDevice::SetTextureQuality** methods.

```
////////////////////////////////////////////////////////////////////
//
// SetRenderState
// Set the render quality and shade information.
//
////////////////////////////////////////////////////////////////////

BOOL
SetRenderState(void)
{
    HRESULT rval;

    // Set the render quality (light toggle, fill mode, shade mode).

    rval = myglobs.dev->lpVtbl->SetQuality(myglobs.dev,
        D3DRMLIGHT_ON | D3DRMFILL_SOLID | D3DRMSHADE_GOURAUD);
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // If you want to change the dithering mode, call SetDither here.

    // If you want a texture quality other than D3DRMTEXTURE_NEAREST
    // (the default value), call SetTextureQuality here.

   return TRUE;
```

```
}
```

The Rendering Loop

The WinMain function calls the RenderLoop function to draw each new frame of the animation. The RenderLoop function performs a few simple tasks:

- Calls the **IDirect3DRMFrame::Move** method to apply the rotations and velocities for all frames in the hierarchy.
- Calls the **IDirect3DRMViewport::Clear** method to clear the current viewport, setting it to the current background color.
- Calls the **IDirect3DRMViewport::Render** method to render the current scene into the current viewport.
- Calls the **IDirect3DRMDevice::Update** method to copy the rendered image to the display.

```
////////////////////////////////////////////////////////////////////
//
// RenderLoop
// Clear the viewport, render the next frame, and update the window.
//
////////////////////////////////////////////////////////////////////

static BOOL
RenderLoop()
{
    HRESULT rval;

    // Tick the scene.

    rval = myglobs.scene->lpVtbl->Move(myglobs.scene, D3DVAL(1.0));
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // Clear the viewport.

    rval = myglobs.view->lpVtbl->Clear(myglobs.view);
    if (rval != D3DRM_OK) {
        return FALSE;
    }

    // Render the scene to the viewport.

    rval = myglobs.view->lpVtbl->Render(myglobs.view, myglobs.scene);
    if (rval != D3DRM_OK) {
        return FALSE;
    }
```

```
    // Update the window.

    rval = myglobs.dev->lpVtbl->Update(myglobs.dev);
    if (rval != D3DRM_OK) {
        return FALSE;
    }
    return TRUE;
}
```

Creating the Scene

After setting up the 3-D environment—choosing a device driver, creating the 3-D device and viewport, setting the rendering state, and so on—Helworld.c calls a series of functions to populate this 3-D environment with objects, frames, and lights:

- *The MyScene Function*
- *The MakeMyFrames Function*
- *The MakeMyLights Function*
- *The SetMyPositions Function*
- *The MakeMyMesh Function*
- *The MakeMyWrap Function*
- *The AddMyTexture Function*

The MyScene Function

The MyScene function in Helworld.c corresponds to the BuildScene function that is implemented in all the Direct3D samples in the DirectX SDK. This is where all the work occurs that displays unique objects with unique textures and lighting effects.

The MyScene function calls a series of locally defined functions that set up the separate features of the scene that is being created. These functions are:

- *MakeMyFrames*
- *MakeMyLights*
- *SetMyPositions*
- *MakeMyMesh*
- *MakeMyWrap*
- *AddMyTexture*

When these functions have set up the visual object, MyScene calls the **IDirect3DRMFrame::AddVisual** method to add the object to the environment's

world frame. After adding the visual object, MyScene no longer needs the interfaces it has created, so it calls the **Release** method repeatedly to release them all.

```
////////////////////////////////////////////////////////////////////
//
// MyScene
// Calls the functions that create the frames, lights, mesh, and
// texture. Releases all interfaces on completion.
//
////////////////////////////////////////////////////////////////////

BOOL
MyScene(LPDIRECT3DRMDEVICE dev, LPDIRECT3DRMVIEWPORT view,
    LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera)
{
    LPDIRECT3DRMFRAME lpLightframe1 = NULL;
    LPDIRECT3DRMFRAME lpWorld_frame = NULL;
    LPDIRECT3DRMLIGHT lpLight1      = NULL;
    LPDIRECT3DRMLIGHT lpLight2      = NULL;
    LPDIRECT3DRMTEXTURE lpTex       = NULL;
    LPDIRECT3DRMWRAP lpWrap         = NULL;
    LPDIRECT3DRMMESHBUILDER lpSphere3_builder = NULL;

    MakeMyFrames(lpScene, lpCamera, &lpLightframe1, &lpWorld_frame);
    MakeMyLights(lpScene, lpCamera, lpLightframe1, &lpLight1,
        &lpLight2);
    SetMyPositions(lpScene, lpCamera, lpLightframe1, lpWorld_frame);
    MakeMyMesh(&lpSphere3_builder);
    MakeMyWrap(lpSphere3_builder, &lpWrap);
    AddMyTexture(lpSphere3_builder, &lpTex);

    // If you need to create a material (for example, to create
    // a shiny surface), call CreateMaterial and SetMaterial here.

    // Now that the visual object has been created, add it
    // to the world frame.

    lpWorld_frame->lpVtbl->AddVisual(lpWorld_frame,
        (LPDIRECT3DRMVISUAL) lpSphere3_builder);

    lpLightframe1->lpVtbl->Release(lpLightframe1);
    lpWorld_frame->lpVtbl->Release(lpWorld_frame);
    lpSphere3_builder->lpVtbl->Release(lpSphere3_builder);
    lpLight1->lpVtbl->Release(lpLight1);
    lpLight2->lpVtbl->Release(lpLight2);
    lpTex->lpVtbl->Release(lpTex);
    lpWrap->lpVtbl->Release(lpWrap);

    return TRUE;
```

```
}
```

The MakeMyFrames Function

The MyScene function calls the MakeMyFrames function to create the frames for the directional light and the world frame used in Helworld.c. MakeMyFrames does this work by calling the **IDirect3DRM::CreateFrame** method.

```
//////////////////////////////////////////////////////////////////////
//
// MakeMyFrames
// Create frames used in the scene.
//
//////////////////////////////////////////////////////////////////////

void MakeMyFrames(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME * lplpLightFrame1,
    LPDIRECT3DRMFRAME * lplpWorld_frame)
{
    lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, lpScene, lplpLightFrame1);
    lpD3DRM->lpVtbl->CreateFrame(lpD3DRM, lpScene, lplpWorld_frame);
}
```

The MakeMyLights Function

The MyScene function calls the MakeMyLights function to create the directional and ambient lights used in Helworld.c. MakeMyLights calls the **IDirect3DRM::CreateLightRGB** and **IDirect3DRMFrame::AddLight** methods to create a bright directional light and add it to a light frame, and to create a dim ambient light and add it to the entire scene. (Ambient lights are always associated with an entire scene.)

```
//////////////////////////////////////////////////////////////////////
//
// MakeMyLights
// Create lights used in the scene.
//
//////////////////////////////////////////////////////////////////////

void MakeMyLights(LPDIRECT3DRMFRAME lpScene, LPDIRECT3DRMFRAME lpCamera,
    LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMLIGHT * lplpLight1, LPDIRECT3DRMLIGHT * lplpLight2)
{
    lpD3DRM->lpVtbl->CreateLightRGB(lpD3DRM, D3DRMLIGHT_DIRECTIONAL,
        D3DVAL(0.9), D3DVAL(0.9), D3DVAL(0.9), lplpLight1);

    lpLightFrame1->lpVtbl->AddLight(lpLightFrame1, *lplpLight1);

    lpD3DRM->lpVtbl->CreateLightRGB(lpD3DRM, D3DRMLIGHT_AMBIENT,
```

```
        D3DVAL(0.1), D3DVAL(0.1), D3DVAL(0.1), lplpLight2);

    lpScene->lpVtbl->AddLight(lpScene, *lplpLight2);
}
```

The SetMyPositions Function

The MyScene function calls the SetMyPositions function to set the positions and orientations of the frames used in Helworld.c. SetMyPositions does this work by calling the **IDirect3DRMFrame::SetPosition** and **IDirect3DRMFrame::SetOrientation** methods. The **IDirect3DRMFrame::SetRotation** method imparts a spin to the frame to which the sphere will be added.

```
////////////////////////////////////////////////////////////////////
//
// SetMyPositions
// Set the positions and orientations of the light, camera, and
// world frames. Establish a rotation for the globe.
//
////////////////////////////////////////////////////////////////////

void SetMyPositions(LPDIRECT3DRMFRAME lpScene,
    LPDIRECT3DRMFRAME lpCamera, LPDIRECT3DRMFRAME lpLightFrame1,
    LPDIRECT3DRMFRAME lpWorld_frame)
{

    lpLightFrame1->lpVtbl->SetPosition(lpLightFrame1, lpScene,
        D3DVAL(2), D3DVAL(0.0), D3DVAL(22));

    lpCamera->lpVtbl->SetPosition(lpCamera, lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0));
    lpCamera->lpVtbl->SetOrientation(lpCamera, lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1),
        D3DVAL(0.0), D3DVAL(1), D3DVAL(0.0));

    lpWorld_frame->lpVtbl->SetPosition(lpWorld_frame, lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(15));
    lpWorld_frame->lpVtbl->SetOrientation(lpWorld_frame, lpScene,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1),
        D3DVAL(0.0), D3DVAL(1), D3DVAL(0.0));

    lpWorld_frame->lpVtbl->SetRotation(lpWorld_frame, lpScene,
        D3DVAL(0.0), D3DVAL(0.1), D3DVAL(0.0), D3DVAL(0.05));
}
```

The MakeMyMesh Function

The MyScene function calls the MakeMyMesh function to load and set the spherical mesh used in Helworld.c. MakeMyMesh calls the **IDirect3DRM::CreateMeshBuilder** method to create an *IDirect3DRMMeshBuilder* interface. Then it calls the **IDirect3DRMMeshBuilder::Load**, **IDirect3DRMMeshBuilder::Scale**, and **IDirect3DRMMeshBuilder::SetColorRGB** methods to prepare the mesh represented by the Sphere3.x file. (The Sphere3.x file is included in the DirectX SDK, as part of the media provided for use with the sample code.)

```
////////////////////////////////////////////////////////////////////
//
// MakeMyMesh
// Create MeshBuilder object, load, scale, and color the mesh.
//
////////////////////////////////////////////////////////////////////

void MakeMyMesh(LPDIRECT3DRMMESHBUILDER * lplpSphere3_builder)
{
    lpD3DRM->lpVtbl->CreateMeshBuilder(lpD3DRM, lplpSphere3_builder);

    (*lplpSphere3_builder)->lpVtbl->Load(*lplpSphere3_builder,
        "sphere3.x", NULL, D3DRMLOAD_FROMFILE, NULL, NULL);

    (*lplpSphere3_builder)->lpVtbl->Scale(*lplpSphere3_builder,
        D3DVAL(2), D3DVAL(2), D3DVAL(2));

    // Set sphere to white to avoid unexpected texture-blending results.

    (*lplpSphere3_builder)->lpVtbl->SetColorRGB(*lplpSphere3_builder,
        D3DVAL(1), D3DVAL(1), D3DVAL(1));
}
```

The MakeMyWrap Function

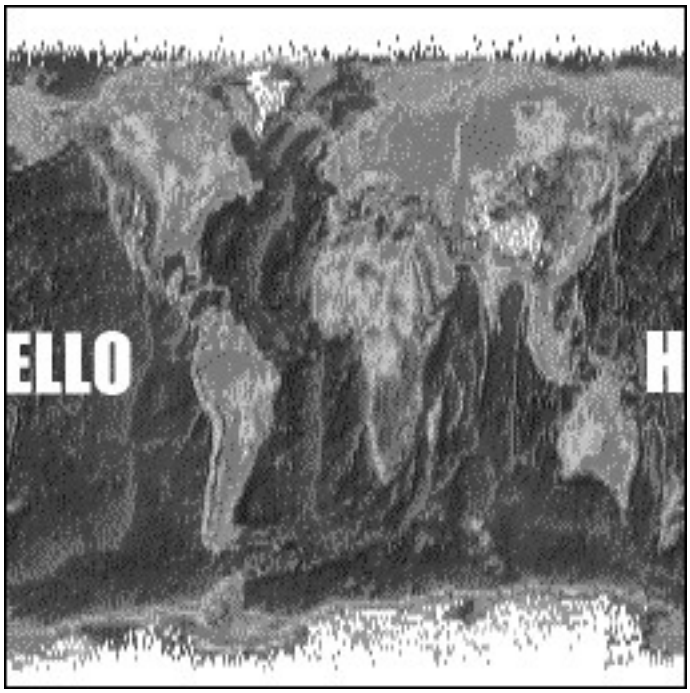
The MyScene function calls the MakeMyWrap function to create and apply texture coordinates to the sphere loaded by the MakeMyMesh function. MakeMyWrap calls the **IDirect3DRMMeshBuilder::GetBox** method to retrieve the bounding box that contains the sphere, and uses the dimensions of that bounding box in a call to the **IDirect3DRM::CreateWrap** method, which creates a cylindrical texture wrap and retrieves the *IDirect3DRMWrap* interface. A call to the **IDirect3DRMWrap::Apply** method applies the texture coordinates to the sphere.

```
////////////////////////////////////////////////////////////////////
//
// MakeMyWrap
// Creates and applies wrap for texture.
```

```
//
//////////////////////////////////////////////////////////////////////

void MakeMyWrap(LPDIRECT3DRMMESHBUILDER sphere3_builder,
                LPDIRECT3DRMWRAP * lpWrap)
{
    D3DVALUE miny, maxy, height;
    D3DRMBOX box;

    sphere3_builder->lpVtbl->GetBox(sphere3_builder, &box);

    maxy = box.max.y;
    miny = box.min.y;
    height = maxy - miny;

    lpD3DRM->lpVtbl->CreateWrap
        (lpD3DRM, D3DRMWRAP_CYLINDER, NULL,
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0),
        D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0),
        D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),
        D3DVAL(0.0), D3DDivide(miny, height),
        D3DVAL(1.0), D3DDivide(-D3DVAL(1.0), height),
        lpWrap);

    (*lpWrap)->lpVtbl->Apply(*lpWrap, (LPDIRECT3DRMOBJECT)
        sphere3_builder);

}
```

The AddMyTexture Function

The MyScene function calls the AddMyTexture function to load a texture and associate it with the sphere. AddMyTexture calls the **IDirect3DRM::LoadTexture** method to load a bitmap called tutor.bmp, and then it calls the **IDirect3DRMMeshBuilder::SetTexture** method to associate the bitmap with the sphere.




```
//////////////////////////////////////////////////////////////////////
//
// AddMyTexture
// Creates and applies wrap for texture.
//
//////////////////////////////////////////////////////////////////////

void AddMyTexture(LPDIRECT3DRMMESHBUILDER lpSphere3_builder,
    LPDIRECT3DRMTEXTURE * lplpTex)
{
    lpD3DRM->lpVtbl->LoadTexture(lpD3DRM, "tutor.bmp", lplpTex);

    // If you need a color depth other than the default (16),
    // call IDirect3DRMTexture::SetShades here.

    lpSphere3_builder->lpVtbl->SetTexture(lpSphere3_builder, *lplpTex);

}
```

Cleaning Up

Helworld.c calls the CleanUp function when it receives a WM_DESTROY message or after several consecutive unsuccessful attempts to call the RenderLoop function.

```
//////////////////////////////////////////////////////////////////////
//
// CleanUp
// Release all D3DRM objects and set the bQuit flag.
//
//////////////////////////////////////////////////////////////////////

void
CleanUp(void)
{
    myglobs.bInitialized = FALSE;
    myglobs.scene->lpVtbl->Release(myglobs.scene);
    myglobs.camera->lpVtbl->Release(myglobs.camera);
    myglobs.view->lpVtbl->Release(myglobs.view);
    myglobs.dev->lpVtbl->Release(myglobs.dev);
    lpD3DRM->lpVtbl->Release(lpD3DRM);
    lpDDClipper->lpVtbl->Release(lpDDClipper);

    myglobs.bQuit = TRUE;
}
```

CHAPTER 2

Direct3D Retained Mode: Reference

Functions

Direct3DRMCreate

Creates an instance of a Direct3DRM object.

HRESULT Direct3DRMCreate(
LPDIRECT3DRM FAR * *lplpD3DRM*
);

Parameters

lplpD3DRM
Address of a pointer that will be initialized with a valid Direct3DRM pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

Direct3DRMObject

D3DRMColorGetAlpha

Retrieves the alpha component of a color.

D3DVALUE D3DRMColorGetAlpha(
D3DCOLOR *d3drmc*
);

Parameters *d3drmc*
Color from which the alpha component is retrieved.

Return Values Returns the alpha value if successful, or zero otherwise.

See Also **D3DRMColorGetBlue**, **D3DRMColorGetGreen**, **D3DRMColorGetRed**

D3DRMColorGetBlue

Retrieves the blue component of a color.

D3DVALUE D3DRMColorGetBlue(
D3DCOLOR *d3drmc*
);

Parameters *d3drmc*
Color from which the blue component is retrieved.

Return Values Returns the blue value if successful, or zero otherwise.

See Also **D3DRMColorGetAlpha**, **D3DRMColorGetGreen**, **D3DRMColorGetRed**

D3DRMColorGetGreen

Retrieves the green component of a color.

D3DVALUE D3DRMColorGetGreen(
D3DCOLOR *d3drmc*
);

Parameters *d3drmc*
Color from which the green component is retrieved.

Return Values Returns the green value if successful, or zero otherwise.

See Also **D3DRMColorGetAlpha**, **D3DRMColorGetBlue**, **D3DRMColorGetRed**

D3DRMColorGetRed

Retrieves the red component of a color.

D3DVALUE D3DRMColorGetRed(
D3DCOLOR *d3drmc*
);

Parameters

d3drmc
Color from which the red component is retrieved.

Return Values Returns the red value if successful, or zero otherwise.

See Also **D3DRMColorGetAlpha**, **D3DRMColorGetBlue**, **D3DRMColorGetGreen**

D3DRMCreateColorRGB

Creates an RGB color from supplied red, green, and blue components.

D3DCOLOR D3DRMCreateColorRGB(
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*
);

Parameters

red, *green*, and *blue*
Components of the RGB color.

Return Values Returns the new RGB value if successful, or zero otherwise.

See Also **D3DRMCreateColorRGBA**

D3DRMCreateColorRGBA

Creates an RGBA color from supplied red, green, blue, and alpha components.

D3DCOLOR D3DRMCreateColorRGBA(
D3DVALUE *red*,
D3DVALUE *green*,

D3DVALUE *blue*,
D3DVALUE *alpha*
);

Parameters

red, *green*, *blue*, and *alpha*
Components of the RGBA color.

Return Values

Returns the new RGBA value if successful, or zero otherwise.

See Also

D3DRMCreateColorRGB

D3DRMFREEFUNCTION

Frees memory. This function is application-defined.

```
typedef VOID (*D3DRMFREEFUNCTION)(LPVOID lpArg);
typedef D3DRMFREEFUNCTION *LPD3DRMFREEFUNCTION;
```

Parameters

lpArg
Address of application-defined data.

Return Values

No return value.

Remarks

Applications might define their own memory-freeing function if the standard C run-time routines do not meet their requirements.

D3DRMMALLOCFUNCTION

Allocates memory. This function is application-defined.

```
typedef LPVOID (*D3DRMMALLOCFUNCTION)(DWORD dwSize);
typedef D3DRMMALLOCFUNCTION *LPD3DRMMALLOCFUNCTION;
```

Parameters

dwSize
Specifies the size, in bytes, of the memory that will be allocated.

Return Values

Returns the address of the allocated memory if successful, or zero otherwise.

Remarks

Applications might define their own memory-allocation function if the standard C run-time routines do not meet their requirements.

D3DRMMatrixFromQuaternion

Calculates the matrix for the rotation that a unit quaternion represents.

void D3DRMMatrixFromQuaternion (
D3DRMMATRIX4D *mat*,
LPD3DRMQUATERNION *lpquat*
);

Parameters

mat
Address that will contain the calculated matrix when the function returns. (The **D3DRMMATRIX4D** type is an array.)

lpquat
Address of the **D3DRMQUATERNION** structure.

Return Values

No return value.

D3DRMQuaternionFromRotation

Retrieves a unit quaternion that represents a rotation of a specified number of radians around the given axis.

LPD3DRMQUATERNION D3DRMQuaternionFromRotation(
LPD3DRMQUATERNION *lpquat*,
LPD3DVECTOR *lpv*,
D3DVALUE *theta*
);

Parameters

lpquat
Address of a **D3DRMQUATERNION** structure that will contain the result of the operation.

lpv
Address of a **D3DVECTOR** structure specifying the axis of rotation.

theta
Number of radians to rotate around the axis specified by the *lpv* parameter.

Return Values

Returns the address of the unit quaternion that was passed as the first parameter if successful, or zero otherwise.

D3DRMQuaternionMultiply

Calculates the product of two quaternion structures.

LPD3DRMQUATERNION D3DRMQuaternionMultiply(
LPD3DRMQUATERNION *lpq*,
LPD3DRMQUATERNION *lpa*,
LPD3DRMQUATERNION *lpb*
);

Parameters

lpq
Address of the **D3DRMQUATERNION** structure that will contain the product of the multiplication.

lpa and *lpb*
Addresses of the **D3DRMQUATERNION** structures that will be multiplied together.

Return Values

Returns the address of the quaternion that was passed as the first parameter if successful, or zero otherwise.

D3DRMQuaternionSlerp

Interpolates between two quaternion structures, using spherical linear interpolation.

LPD3DRMQUATERNION D3DRMQuaternionSlerp(
LPD3DRMQUATERNION *lpq*,
LPD3DRMQUATERNION *lpa*,
LPD3DRMQUATERNION *lpb*,
D3DVALUE *alpha*
);

Parameters

lpq
Address of the **D3DRMQUATERNION** structure that will contain the interpolation.

lpa and *lpb*
Addresses of the **D3DRMQUATERNION** structures that are used as the starting and ending points for the interpolation, respectively.

alpha
Value between 0 and 1 that specifies how far to interpolate between *lpa* and *lpb*.

Return Values Returns the address of the quaternion that was passed as the first parameter if successful, or zero otherwise.

D3DRMREALLOCFUNCTION

Reallocates memory. This function is application-defined.

```
typedef LPVOID (*D3DRMREALLOCFUNCTION)(LPVOID lpArg,
    DWORD dwSize);
typedef D3DRMREALLOCFUNCTION *LPD3DRMREALLOCFUNCTION;
```

Parameters

lpArg
Address of application-defined data.

dwSize
Size, in bytes, of the reallocated memory.

Return Values Returns an address of the reallocated memory if successful, or zero otherwise.

Remarks Applications may define their own memory-reallocation function if the standard C run-time routines do not meet their requirements.

D3DRMVectorAdd

Adds two vectors.

LPD3DVECTOR D3DRMVectorAdd(
LPD3DVECTOR *lpd*,
LPD3DVECTOR *lps1*,
LPD3DVECTOR *lps2*
);

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain the result of the addition.

lps1 and *lps2*
Addresses of the **D3DVECTOR** structures that will be added together.

Return Values Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

D3DRMVectorCrossProduct

Calculates the cross product of the two vector arguments.

LPD3DVECTOR D3DRMVectorCrossProduct(
LPD3DVECTOR *lpd*,
LPD3DVECTOR *lps1*,
LPD3DVECTOR *lps2*
);

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain the result of the cross product.

lps1 and *lps2*
Addresses of the **D3DVECTOR** structures from which the cross product is produced.

Return Values

Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

D3DRMVectorDotProduct

Returns the vector dot product.

D3DVALUE D3DRMVectorDotProduct(
LPD3DVECTOR *lps1*,
LPD3DVECTOR *lps2*
);

Parameters

lps1 and *lps2*
Addresses of the **D3DVECTOR** structures from which the dot product is produced.

Return Values

Returns the result of the dot product if successful, or zero otherwise.

D3DRMVectorModulus

Returns the length of a vector according to the following formula:

$$length = \sqrt{x^2 + y^2 + z^2}$$

D3DVALUE D3DRMVectorModulus(
LPD3DVECTOR *lpv*
);

Parameters

lpv
Address of the **D3DVECTOR** structure whose length is returned.

Return Values

Returns the length of the **D3DVECTOR** structure if successful, or zero otherwise.

D3DRMVectorNormalize

Scales a vector so that its modulus is 1.

LPD3DVECTOR D3DRMVectorNormalize(
LPD3DVECTOR *lpv*
);

Parameters

lpv
Address of a **D3DVECTOR** structure that will contain the result of the scaling operation.

Return Values

Returns the address of the vector that was passed as the first parameter if successful, or zero if an error occurs, such as if, for example, a zero vector was passed.

D3DRMVectorRandom

Returns a random unit vector.

LPD3DVECTOR D3DRMVectorRandom(
LPD3DVECTOR *lpd*
);

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain a random unit vector.

Return Values Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

D3DRMVectorReflect

Reflects a ray about a given normal.

LPD3DVECTOR D3DRMVectorReflect(
LPD3DVECTOR *lpd*,
LPD3DVECTOR *lpRay*,
LPD3DVECTOR *lpNorm*
);

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain the result of the operation.

lpRay
Address of a **D3DVECTOR** structure that will be reflected about a normal.

lpNorm
Address of a **D3DVECTOR** structure specifying the normal about which the vector specified in *lpRay* is reflected.

Return Values Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

D3DRMVectorRotate

Rotates a vector around a given axis. Returns a normalized vector if successful.

LPD3DVECTOR D3DRMVectorRotate(
LPD3DVECTOR *lpr*,
LPD3DVECTOR *lpv*,
LPD3DVECTOR *lpaxis*,
D3DVALUE *theta*
);

Parameters

lpr
Address of a **D3DVECTOR** structure that will contain the normalized result of the operation.

lpv
Address of a **D3DVECTOR** structure that will be rotated around the given axis.

lpaxis
Address of a **D3DVECTOR** structure that is the axis of rotation.

theta
The rotation in radians.

Return Values Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise. The vector is normalized.

D3DRMVectorScale

Scales a vector uniformly in all three axes.

LPD3DVECTOR D3DRMVectorScale(
LPD3DVECTOR *lpd*,
LPD3DVECTOR *lps*,
D3DVALUE *factor*
);

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain the result of the operation.

lps
Address of a **D3DVECTOR** structure that this function scales.

factor
Scaling factor. A value of 1 does not change the scaling; a value of 2 doubles it, and so on.

Return Values Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

D3DRMVectorSubtract

Subtracts two vectors.

LPD3DVECTOR D3DRMVectorSubtract(
LPD3DVECTOR *lpd*,
LPD3DVECTOR *lps1*,
LPD3DVECTOR *lps2*

```
);
```

Parameters

lpd
Address of a **D3DVECTOR** structure that will contain the result of the operation.

lps1
Address of the **D3DVECTOR** structure from which *lps2* is subtracted.

lps2
Address of the **D3DVECTOR** structure that is subtracted from *lps1*.

Return Values

Returns the address of the vector that was passed as the first parameter if successful, or zero otherwise.

Callback Functions

D3DRMDEVICEPALETTECALLBACK

Enumerates palette entries. This callback function is application-defined.

```
void (
    *D3DRMDEVICEPALETTECALLBACK
    )
(
    LPDIRECT3DRMDEVICE lpDirect3DRMDev,
    LPVOID lpArg,
    DWORD dwIndex,
    LONG red,
    LONG green,
    LONG blue
    );
```

Parameters

lpDirect3DRMDev
Address of the *IDirect3DRMDevice* interface for this device.

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

dwIndex
Index of the palette entry being described.

red, *green*, and *blue*
Red, green, and blue components of the color at the given index in the palette.

Return Values

No return value.

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.

D3DRMFRAMEMOVECALLBACK

Enables an application to apply customized algorithms when a frame is moved or updated. You can use this callback function to compensate for changing frame rates. This callback function is application-defined.

```
void (
    *D3DRMFRAMEMOVECALLBACK
    )(
    LPDIRECT3DRMFRAME lpD3DRMFrame,
    LPVOID lpArg,
    D3DVALUE delta
    );
```

Parameters

lpD3DRMFrame
Address of the Direct3DRMFrame object that is being moved.

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

delta
Amount of change to apply to the movement. There are two components to the change in position of a frame: linear and rotational. The change in each component is equal to *velocity_of_component* × *delta*. Although either or both of these velocities can be set relative to any frame, the system automatically converts them to velocities relative to the parent frame for the purpose of applying time deltas.

Return Values

No return value.

Remarks

Your application can synthesize the acceleration of a frame relative to its parent frame. To do so, on each tick your application should set the velocity of the child frame relative to itself to (*a* units per tick) × 1 tick, where *a* is the required acceleration. This is equal to *a* × *delta* units per tick. Internally, *a* × *delta* units per tick relative to the child frame is converted to (*v* + (*a* × *delta*)) units per tick relative to the parent frame, where *v* is the current velocity of the child relative to the parent.

You can add and remove this callback function from your application by using the **IDirect3DRMFrame::AddMoveCallback** and **IDirect3DRMFrame::DeleteMoveCallback** methods.

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.

D3DRMLOADCALLBACK

Loads objects named in a call to the **IDirect3DRM::Load** method. This callback function is application-defined.

void (
**D3DRMLOADCALLBACK*
)(
LPDIRECT3DRMOBJECT *lpObject*,
REFIID *ObjectGuid*,
LPVOID *lpArg*
);

Parameters

lpObject
Address of the Direct3DRMObject being loaded.

ObjectGuid
Globally unique identifier (GUID) of the object being loaded.

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

Return Values

No return value.

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.

See Also

IDirect3DRM::Load

D3DRMLOADTEXTURECALLBACK

Loads texture maps from a file or resource named in a call to one of the **Load** methods. This callback function is application-defined.

HRESULT (
**D3DRMLOADTEXTURECALLBACK*
)(

char **tex_name*,
void **lpArg*,
LPDIRECT3DRMTEXTURE **lpD3DRMTex*
);

Parameters

tex_name
Address of a string containing the name of the texture.

lpArg
Address of application-specific data.

lpD3DRMTex
Address of the Direct3DRMTexture object.

Return Values

Should return D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Applications can use this callback function to implement support for textures that are not in the Windows bitmap (.bmp) or Portable Pixmap (.ppm) P6 format.

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.

See Also

IDirect3DRM::Load, **IDirect3DRMAnimationSet::Load**, **IDirect3DRMFrame::Load**, **IDirect3DRMMeshBuilder::Load**

D3DRMOBJECTCALLBACK

Enumerates objects in response to a call to the **IDirect3DRM::EnumerateObjects** method. This callback function is application-defined.

void (
**D3DRMOBJECTCALLBACK*
)(
LPDIRECT3DRMOBJECT *lpD3DRMobj*,
LPVOID *lpArg*
);

Parameters

lpD3DRMobj
Address of an *IDirect3DRMObject* interface for the object being enumerated. The application must call the **Release** method for each enumerated object.

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

Return Values No return value.

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.

See Also **IDirect3DRM::EnumerateObjects**

D3DRMUPDATECALLBACK

Alerts the application whenever the device changes. This callback function is application-defined.

void (
**D3DRMUPDATECALLBACK*
)(
LPDIRECT3DRMDEVICE *lpobj*,
LPVOID *lpArg*,
int *iRectCount*,
LPD3DRECT *d3dRectUpdate*
);

Parameters

lpobj
Address of the Direct3DRMDevice object to which this callback function applies.

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

iRectCount
Number of rectangles specified in the *d3dRectUpdate* parameter.

d3dRectUpdate
Array of one or more **D3DRECT** structures that describe the area to be updated. The coordinates are specified in device units.

Return Values No return value.

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.

See Also **IDirect3DRMDevice::AddUpdateCallback**, **IDirect3DRMDevice::DeleteUpdateCallback**, **IDirect3DRMDevice::Update**

D3DRMUSERVISUALCALLBACK

Alerts an application that supplies user-visual objects that it should execute the execute buffer. This function is application-defined.

```
int (
    *D3DRMUSERVISUALCALLBACK
    )(
    LPDIRECT3DRMUSERVISUAL lpD3DRMUV,
    LPVOID lpArg,
    D3DRMUSERVISUALREASON lpD3DRMUVreason,
    LPDIRECT3DRMDEVICE lpD3DRMDev,
    LPDIRECT3DRMVIEWPORT lpD3DRMview
    );
```

Parameters

lpD3DRMUV
Address of the Direct3DRMUserVisual object.

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

lpD3DRMUVreason
One of the members of the **D3DRMUSERVISUALREASON** enumerated type:

D3DRMUSERVISUAL_CANSEE
The application should return TRUE if the user-visual object is visible in the viewport. In this case, the application uses the device specified in the *lpD3DRMview* parameter.

D3DRMUSERVISUAL_RENDER
The application should render the user-visual element. In this case, the application uses the device specified in the *lpD3DRMDev* parameter.

lpD3DRMDev
Address of a Direct3DRMDevice object used to render the Direct3DRMUserVisual object.

lpD3DRMview
Address of a Direct3DRMViewport object used to determine whether the Direct3DRMUserVisual object is visible.

Return Values

Returns TRUE if the *lpD3DRMUVreason* parameter is D3DRMUSERVISUAL_CANSEE and the user-visual object is visible in the viewport. Returns FALSE otherwise. If the *lpD3DRMUVreason* parameter is D3DRMUSERVISUAL_RENDER, the return value is application-defined. It is always safe to return TRUE.

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.

See Also **IDirect3DRMUserVisual::Init**

D3DRMWRAPCALLBACK

This callback function is not supported.

void (
**D3DRMWRAPCALLBACK*
)(
LPD3DVECTOR *lpD3DVector*,
int* *lpU*,
int* *lpV*,
LPD3DVECTOR *lpD3DRMVA*,
LPD3DVECTOR *lpD3DRMVB*,
LPVOID *lpArg*
);

IDirect3DRM Array Interfaces

The array interfaces make it possible for your application to group objects into arrays, making it simpler to apply operations to the entire group. The following array interfaces are available:

IDirect3DRMArray

IDirect3DRMDeviceArray

IDirect3DRMFaceArray

IDirect3DRMFrameArray

IDirect3DRMLightArray

IDirect3DRMObjectArray

IDirect3DRMPickedArray

IDirect3DRMPicked2Array

IDirect3DRMViewportArray

IDirect3DRMVisualArray

IDirect3DRMArray

The **IDirect3DRMArray** interface organizes groups of objects. Applications typically use array objects that are subsidiary to this interface, rather than using this interface directly. This section is a reference to the methods of this interface.

The **IDirect3DRMArray** interface supports the **GetSize** method.

The **IDirect3DRMArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. This interface supports the following three methods:

AddRef

QueryInterface

Release

IDirect3DRMArray::GetSize

Retrieves the size, in objects, of the Direct3DRMArray object.

DWORD GetSize( );

Return Values Returns the size.

IDirect3DRMDeviceArray

Applications use the methods of the **IDirect3DRMDeviceArray** interface to organize device objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMDevice, IDirect3DRMDevice2, and IDirect3DRMDeviceArray Interfaces*.

The **IDirect3DRMDeviceArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMDeviceArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMDeviceArray object is obtained by calling the **IDirect3DRM::GetDevices** method.

IDirect3DRMDeviceArray::GetElement

Retrieves a specified element in a Direct3DRMDeviceArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMDEVICE * *lplpD3DRMDevice*
);

Parameters

index
Element in the array.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDeviceArray::GetSize

Retrieves the number of elements contained in a Direct3DRMDeviceArray object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRMFaceArray

Applications use the methods of the **IDirect3DRMFaceArray** interface to organize faces in a mesh. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMFace and IDirect3DRMFaceArray Interfaces*.

The **IDirect3DRMFaceArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMFaceArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMFaceArray object is obtained by calling the **IDirect3DRMMeshBuilder::GetFaces** method.

IDirect3DRMFaceArray::GetElement

Retrieves a specified element in a Direct3DRMFaceArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMFACE * *lplpD3DRMFace*
);

Parameters

index
Element in the array.

lplpD3DRMFace
Address that will be filled with a pointer to an *IDirect3DRMFace* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFaceArray::GetSize

Retrieves the number of elements contained in a Direct3DRMFaceArray object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRMFrameArray

Applications use the methods of the **IDirect3DRMFrameArray** interface to organize frame objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMFrame, IDirect3DRMFrame2, and IDirect3DRMFrameArray Interfaces*.

The **IDirect3DRMFrameArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMFrameArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMFrameArray object is obtained by calling the **IDirect3DRMPickedArray::GetPick**, **IDirect3DRMPicked2Array::GetPick**, or **IDirect3DRMFrame::GetChildren** method.

IDirect3DRMFrameArray::GetElement

Retrieves a specified element in a Direct3DRMFrameArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMFRAME * *lplpD3DRMFrame*
);

Parameters

index
Element in the array.

lplpD3DRMFrame
Address that will be filled with a pointer to an *IDirect3DRMFrame* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrameArray::GetSize

Retrieves the number of elements contained in a Direct3DRMFrameArray object.

DWORD GetSize( );

Return Values Returns the number of elements.

IDirect3DRMLightArray

Applications use the methods of the **IDirect3DRMLightArray** interface to organize light objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMLight and IDirect3DRMLightArray Interfaces*.

The **IDirect3DRMLightArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMLightArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMLightArray object is obtained by calling the **IDirect3DRMFrame::GetLights** method.

IDirect3DRMLightArray::GetElement

Retrieves a specified element in a Direct3DRMLightArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMLIGHT * *lplpD3DRMLight*
);

Parameters

index
Element in the array.

lplpD3DRMLight
Address that will be filled with a pointer to an *IDirect3DRMLight* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMLightArray::GetSize

Retrieves the number of elements contained in a Direct3DRMLightArray object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRMObjectArray

Applications use the methods of the **IDirect3DRMObjectArray** interface to organize Direct3DRMObject objects. This section is a reference to the methods of this interface. For a conceptual overview, see *Direct3DRMObject*.

The **IDirect3DRMObjectArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMObjectArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMObjectArray object is obtained by calling the **IDirect3DRMInterpolator::GetAttachedObjects** method.

IDirect3DRMObjectArray::GetElement

Retrieves a specified element in a Direct3DRMObjectArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMOBJECT * *lplpD3DRMObject*
);

Parameters

index
Element in the array.

lplpD3DRMObject
Address that will be filled with a pointer to an *IDirect3DRMObject* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObjectArray::GetSize

Retrieves the number of elements contained in a Direct3DRMObjectArray object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRMPickedArray

Applications use the methods of the **IDirect3DRMPickedArray** interface to organize pick objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMPickedArray and IDirect3DRMPicked2Array Interfaces*.

The **IDirect3DRMPickedArray** interface supports the following methods:

GetPick

GetSize

The **IDirect3DRMPickedArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMPickedArray object is obtained by calling the **IDirect3DRMViewport::Pick** method.

IDirect3DRMPickedArray::GetPick

Retrieves the Direct3DRMVisual and Direct3DRMFrame objects intersected by the specified pick.

HRESULT GetPick(
DWORD *index*,
LPDIRECT3DRMVISUAL * *lplpVisual*,
LPDIRECT3DRMFRAMEARRAY * *lplpFrameArray*,
LPD3DRMPICKDESC *lpD3DRMPickDesc*
);

Parameters

index
Index into the pick array identifying the pick for which information will be retrieved.

lplpVisual
Address that will contain a pointer to the Direct3DRMVisual object associated with the specified pick.

lplpFrameArray
Address that will contain a pointer to the Direct3DRMFrameArray object associated with the specified pick.

lpD3DRMPickDesc
Address of a **D3DRMPICKDESC** structure specifying the pick position and face and group identifiers of the objects being retrieved.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMViewport::Pick

IDirect3DRMPickedArray::GetSize

Retrieves the number of elements contained in a Direct3DRMPickedArray object.

DWORD GetSize( );

Return Values Returns the number of elements.

IDirect3DRMPicked2Array

Applications use the methods of the **IDirect3DRMPicked2Array** interface to organize pick objects and return more information about the pick objects. The **D3DRMPICKDESC2** structure returned by **IDirect3DRMPicked2Array::GetPick** contains the face and group identifiers, pick position, horizontal and vertical texture coordinates for the vertex, vertex normal, and color of the intersected objects. A pointer to this interface pointer is returned in the **IDirect3DRMFrame2::RayPick** method during ray picks.

This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMPickedArray and IDirect3DRMPicked2Array Interfaces*.

The **IDirect3DRMPicked2Array** interface supports the following methods:

GetPick

GetSize

The **IDirect3DRMPicked2Array** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMPicked2Array object is obtained by calling the **IDirect3DRMFrame2::RayPick** method.

IDirect3DRMPicked2Array::GetPick

Retrieves the Direct3DRMVisual and Direct3DRMFrame objects intersected by the specified pick.

HRESULT GetPick(
DWORD *index*,
LPDIRECT3DRMVISUAL * *lplpVisual*,
LPDIRECT3DRMFRAMEARRAY * *lplpFrameArray*,
LPD3DRMPICKDESC2 *lpD3DRMPickDesc2*
);

Parameters

index
Index into the pick array identifying the pick for which information will be retrieved.

lplpVisual
Address that will contain a pointer to the Direct3DRMVisual object associated with the specified pick.

lplpFrameArray
Address that will contain a pointer to the Direct3DRMFrameArray object associated with the specified pick.

lpD3DRMPickDesc
Address of a **D3DRMPICKDESC2** structure specifying the face and group identifiers, pick position, horizontal and vertical texture coordinates for the vertex, vertex normal, and color of the intersected objects.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::RayPick

IDirect3DRMPicked2Array::GetSize

Retrieves the number of elements contained in a Direct3DRMPicked2Array object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRMViewportArray

Applications use the methods of the **IDirect3DRMViewportArray** interface to organize viewport objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMViewport and IDirect3DRMViewportArray Interface*.

The **IDirect3DRMViewportArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMViewportArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMViewportArray object is obtained by calling the **IDirect3DRM::CreateFrame** method.

IDirect3DRMViewportArray::GetElement

Retrieves a specified element in a Direct3DRMViewportArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMVIEWPORT * *lplpD3DRMViewport*
);

Parameters

index
Element in the array.

lplpD3DRMViewport
Address that will be filled with a pointer to an *IDirect3DRMViewport* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewportArray::GetSize

Retrieves the number of elements contained in a Direct3DRMViewportArray object.

DWORD GetSize( );

Return Values Returns the number of elements.

IDirect3DRMVisualArray

Applications use the methods of the **IDirect3DRMVisualArray** interface to organize groups of visual objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMVisual and IDirect3DRMVisualArray Interfaces*.

The **IDirect3DRMVisualArray** interface supports the following methods:

GetElement

GetSize

The **IDirect3DRMVisualArray** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMVisualArray object is obtained by calling the **IDirect3DRMFrame::GetVisuals** method.

IDirect3DRMVisualArray::GetElement

Retrieves a specified element in a Direct3DRMVisualArray object.

HRESULT GetElement(
DWORD *index*,
LPDIRECT3DRMVISUAL * *lplpD3DRMVisual*
);

Parameters

index
Element in the array.

lplpD3DRMVisual
Address that will be filled with a pointer to an **IDirect3DRMVisual** interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMVisualArray::GetSize

Retrieves the number of elements contained in a Direct3DRMVisualArray object.

DWORD GetSize( );

Return Values

Returns the number of elements.

IDirect3DRM

Applications use the methods of the **IDirect3DRM** interface to create Direct3DRM objects and work with system-level variables. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRM and IDirect3DRM2 Interfaces*.

The methods of the **IDirect3DRM** interface can be organized into the following groups:

Animation	**CreateAnimation**
	CreateAnimationSet
Devices	**CreateDevice**
	CreateDeviceFromClipper
	CreateDeviceFromD3D
	CreateDeviceFromSurface
	GetDevices
Enumeration	**EnumerateObjects**
Faces	**CreateFace**

Frames	**CreateFrame**
Lights	**CreateLight** **CreateLightRGB**
Materials	**CreateMaterial**
Meshes	**CreateMesh** **CreateMeshBuilder**
Miscellaneous	**CreateObject** **CreateUserVisual** **GetNamedObject** **Load** **Tick**
Search paths	**AddSearchPath** **GetSearchPath** **SetSearchPath**
Shadows	**CreateShadow**
Textures	**CreateTexture** **CreateTextureFromSurface** **LoadTexture** **LoadTextureFromResource** **SetDefaultTextureColors** **SetDefaultTextureShades**
Viewports	**CreateViewport**
Wraps	**CreateWrap**

The **IDirect3DRM** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef
QueryInterface

Release

The **IDirect3DRM** COM interface is created by calling the **Direct3DRMCreate** function.

IDirect3DRM::AddSearchPath

Adds a list of directories to the end of the current file search path.

HRESULT AddSearchPath(
LPCSTR *lpPath*
);

Parameters

lpPath
Address of a null-terminated string specifying the path to add to the current search path.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

For Windows, the path should be a list of directories separated by semicolons (;).

See Also

IDirect3DRM::SetSearchPath

IDirect3DRM::CreateAnimation

Creates an empty Direct3DRMAnimation object.

HRESULT CreateAnimation(
LPDIRECT3DRMANIMATION * *lplpD3DRMAnimation*
);

Parameters

lplpD3DRMAnimation
Address that will be filled with a pointer to an *IDirect3DRMAnimation* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateAnimationSet

Creates an empty Direct3DRMAnimationSet object.

HRESULT CreateAnimationSet (
LPDIRECT3DRMANIMATIONSET * *lplpD3DRMAnimationSet*
);

Parameters

lplpD3DRMAnimationSet
Address that will be filled with a pointer to an *IDirect3DRMAnimationSet* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateDevice

Not implemented on the Windows platform.

HRESULT CreateDevice(
DWORD *dwWidth*,
DWORD *dwHeight*,
LPDIRECT3DRMDEVICE* *lplpD3DRMDevice*
);

IDirect3DRM::CreateDeviceFromClipper

Creates a Direct3DRM Windows device by using a specified DirectDrawClipper object.

HRESULT CreateDeviceFromClipper(
LPDIRECTDRAWCLIPPER *lpDDClipper*,
LPGUID *lpGUID*,
int *width*,
int *height*,
LPDIRECT3DRMDEVICE * *lplpD3DRMDevice*
);

Parameters

lpDDClipper
Address of a DirectDrawClipper object.

lpGUID
Address of a globally unique identifier (GUID). This parameter can be NULL.

width and *height*
Width and height of the device to be created.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the *lpGUID* parameter is NULL, the system searches for a device with a default set of device capabilities. This is the recommended way to create a Retained-Mode device because it always works, even if the user installs new hardware.

The system describes the default settings by using the following flags from the **D3DPRIMCAPS** structure in internal device-enumeration calls:

D3DPCMPCAPS_LESSEQUAL

D3DPMISCCAPS_CULLCCW

D3DPRASTERCAPS_FOGVERTEX

D3DPSHADECAPS_ALPHAFLATSTIPPLED

D3DPTADDRESSCAPS_WRAP

D3DPTBLENDCAPS_COPY | **D3DPTBLENDCAPS_MODULATE**

D3DPTEXTURECAPS_PERSPECTIVE |
D3DPTEXTURECAPS_TRANSPARENCY

D3DPTFILTERCAPS_NEAREST

If a hardware device is not found, the monochromatic (ramp) software driver is loaded. An application should enumerate devices instead of specifying NULL for *lpGUID* if it has special needs that are not met by this list of default settings.

IDirect3DRM::CreateDeviceFromD3D

Creates a Direct3DRM Windows device by using specified Direct3D objects.

HRESULT CreateDeviceFromD3D(
LPDIRECT3D *lpD3D*,
LPDIRECT3DDEVICE *lpD3DDev*,
LPDIRECT3DRMDEVICE * *lplpD3DRMDevice*

);

Parameters

lpD3D
Address of an instance of Direct3D.

lpD3DDev
Address of a Direct3D device object.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateDeviceFromSurface

Creates a Windows device for rendering from the specified DirectDraw surfaces.

HRESULT CreateDeviceFromSurface(
LPGUID *lpGUID*,
LPDIRECTDRAW *lpDD*,
LPDIRECTDRAWSURFACE *lpDDSBack*,
LPDIRECT3DRMDEVICE * *lplpD3DRMDevice*
);

Parameters

lpGUID
Address of the globally unique identifier (GUID) used as the required device driver. If this parameter is NULL, the default device driver is used.

lpDD
Address of the DirectDraw object that is the source of the DirectDraw surface.

lpDDSBack
Address of the DirectDrawSurface object that represents the back buffer.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateFace

Creates an instance of the *IDirect3DRMFace* interface.

HRESULT CreateFace(
LPDIRECT3DRMFACE * *lplpd3drmFace*
);

Parameters

lplpd3drmFace
Address that will be filled with a pointer to an *IDirect3DRMFace* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateFrame

Creates a new child frame of the given parent frame.

HRESULT CreateFrame(
LPDIRECT3DRMFRAME *lpD3DRMFrame***,**
LPDIRECT3DRMFRAME* *lplpD3DRMFrame*
);

Parameters

lpD3DRMFrame
Address of a frame that is to be the parent of the new frame.

lplpD3DRMFrame
Address that will be filled with a pointer to an *IDirect3DRMFrame* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The child frame inherits the motion attributes of its parent. For example, if the parent is moving with a given velocity, the child frame will also move with that velocity. Furthermore, if the parent is set rotating, the child frame will rotate about the origin of the parent. Frames that have no parent are called scenes. To create a scene, specify NULL as the parent. An application can create a frame with no parent and then associate it with a parent frame later by using the **IDirect3DRMFrame::AddChild** method.

See Also

IDirect3DRMFrame::AddChild

IDirect3DRM::CreateLight

Creates a new light source with the given type and color.

HRESULT CreateLight(
D3DRMLIGHTTYPE *d3drmltLightType***,**
D3DCOLOR *cColor***,**
LPDIRECT3DRMLIGHT* *lplpD3DRMLight*
);

Parameters

d3drmltLightType
One of the lighting types given in the **D3DRMLIGHTTYPE** enumerated type.

cColor
Color of the light.

lplpD3DRMLight
Address that will be filled with a pointer to an *IDirect3DRMLight* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateLightRGB

Creates a new light source with the given type and color.

HRESULT CreateLightRGB(
D3DRMLIGHTTYPE *ltLightType***,**
D3DVALUE *vRed***,**
D3DVALUE *vGreen***,**
D3DVALUE *vBlue***,**
LPDIRECT3DRMLIGHT* *lplpD3DRMLight*
);

Parameters

ltLightType
One of the lighting types given in the **D3DRMLIGHTTYPE** enumerated type.

vRed, vGreen, and *vBlue*
Color of the light.

lplpD3DRMLight
Address that will be filled with a pointer to an *IDirect3DRMLight* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateMaterial

Creates a material with the given specular property.

HRESULT CreateMaterial(
D3DVALUE *vPower*,
LPDIRECT3DRMMATERIAL * *lplpD3DRMMaterial*
);

Parameters

vPower
Sharpness of the reflected highlights, with a value of 5 giving a metallic look and higher values giving a more plastic look to the rendered surface.

lplpD3DRMMaterial
Address that will be filled with a pointer to an *IDirect3DRMMaterial* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateMesh

Creates a new mesh object with no faces. The mesh is not visible until it is added to a frame.

HRESULT CreateMesh(
LPDIRECT3DRMMESH* *lplpD3DRMMesh*
);

Parameters

lplpD3DRMMesh
Address that will be filled with a pointer to an *IDirect3DRMMesh* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateMeshBuilder

Creates a new mesh builder object.

HRESULT CreateMeshBuilder(
LPDIRECT3DRMMESHBUILDER* *lplpD3DRMMeshBuilder*
);

Parameters

lplpD3DRMMeshBuilder
Address that will be filled with a pointer to an *IDirect3DRMMeshBuilder* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateObject

Creates a new object without initializing the object.

HRESULT CreateObject(
REFCLSID *rclsid*,
LPUNKNOWN *pUnkOuter*,
REFIID *riid*,
LPVOID FAR* *ppv*
);

Parameters

rclsid
Class identifier for the new object.

pUnkOuter
Allows COM aggregation features.

riid
Interface identifier of the object to be created.

ppv
Address of a pointer to the object when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An application that calls this method must initialize the object that has been created. (The other creation methods of the *IDirect3DRM* interface initialize the object automatically.) To initialize the new object, you should use the **Init** method

for that object. An application should call the **Init** method only once to initialize any given object.

Applications can use this method to implement aggregation in Direct3DRM objects.

IDirect3DRM::CreateShadow

Creates a shadow by using the specified visual and light, projecting the shadow onto the specified plane. The shadow is a visual that should be added to the frame that contains the visual.

HRESULT CreateShadow(
LPDIRECT3DRMVISUAL *lpVisual*,
LPDIRECT3DRMLIGHT *lpLight*,
D3DVALUE *px*,
D3DVALUE *py*,
D3DVALUE *pz*,
D3DVALUE *nx*,
D3DVALUE *ny*,
D3DVALUE *nz*,
LPDIRECT3DRMVISUAL * *lplpShadow*
);

Parameters

lpVisual
Address of the Direct3DRMVisual object that is casting the shadow.

lpLight
Address of the *IDirect3DRMLight* interface that is the light source.

px, py, and *pz*
Plane that the shadow is to be projected on.

nx, ny, and *nz*
Normal to the plane that the shadow is to be projected on.

lplpShadow
Address of a pointer to be initialized with a valid pointer to the shadow visual, if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateTexture

Creates a texture from an image in memory.

HRESULT CreateTexture(
LPD3DRMIMAGE *lpImage*,
LPDIRECT3DRMTEXTURE* *lplpD3DRMTexture*
);

Parameters

lpImage
Address of a **D3DRMIMAGE** structure describing the source for the texture.

lplpD3DRMTexture
Address that will be filled with a pointer to an *IDirect3DRMTexture* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The memory associated with the image is used each time the texture is rendered, rather than the memory being copied into Direct3DRM's buffers. This allows the image to be used both as a rendering target and as a texture.

IDirect3DRM::CreateTextureFromSurface

Creates a texture from a specified DirectDraw surface.

HRESULT CreateTextureFromSurface(
LPDIRECTDRAWSURFACE *lpDDS*,
LPDIRECT3DRMTEXTURE * *lplpD3DRMTexture*
);

Parameters

lpDDS
Address of the DirectDrawSurface object containing the texture.

lplpD3DRMTexture
Address that will be filled with a pointer to an *IDirect3DRMTexture* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateUserVisual

Creates an application-defined visual object, which can then be added to a scene and rendered by using an application-defined handler.

HRESULT CreateUserVisual(
D3DRMUSERVISUALCALLBACK *fn*,
LPVOID *lpArg*,
LPDIRECT3DRMUSERVISUAL * *lplpD3DRMUV*
);

Parameters

fn
Application-defined **D3DRMUSERVISUALCALLBACK** callback function.

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

lplpD3DRMUV
Address that will be filled with a pointer to an *IDirect3DRMUserVisual* interface if the call succeeds.

Return Values

Should return D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::CreateViewport

Creates a viewport on a device with device coordinates (*dwXPos*, *dwYPos*) to (*dwXPos* + *dwWidth*, *dwYPos* + *dwHeight*).

HRESULT CreateViewport(
LPDIRECT3DRMDEVICE *lpDev*,
LPDIRECT3DRMFRAME *lpCamera*,
DWORD *dwXPos*,
DWORD *dwYPos*,
DWORD *dwWidth*,
DWORD *dwHeight*,
LPDIRECT3DRMVIEWPORT* *lplpD3DRMViewport*
);

Parameters

lpDev
Device on which the viewport is to be created.

lpCamera
Frame that describes the position and direction of the view.

dwXPos, dwYPos, dwWidth, and *dwHeight*
Position and size of the viewport, in device coordinates. The viewport size cannot be larger than the physical device or the method will fail.

lplpD3DRMViewport
Address that will be filled with a pointer to an *IDirect3DRMViewport* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. If the viewport size is larger than the physical device, returns D3DRMERR_BADVALUE. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The viewport displays objects in the scene that contains the camera, with the view direction and up vector taken from the camera. The viewport size cannot be larger than the physical device.

IDirect3DRM::CreateWrap

Creates a wrapping function that can be used to assign texture coordinates to faces and meshes. The vector [*ox oy oz*] gives the origin of the wrap, [*dx dy dz*] gives its z-axis, and [*ux uy uz*] gives its y-axis. The 2D vectors [*ou ov*] and [*su sv*] give an origin and scale factor in the texture applied to the result of the wrapping function.

HRESULT CreateWrap(
D3DRMWRAPTYPE *type*,
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *ox*,
D3DVALUE *oy*,
D3DVALUE *oz*,
D3DVALUE *dx*,
D3DVALUE *dy*,
D3DVALUE *dz*,
D3DVALUE *ux*,
D3DVALUE *uy*,
D3DVALUE *uz*,
D3DVALUE *ou*,
D3DVALUE *ov*,
D3DVALUE *su*,
D3DVALUE *sv*,
LPDIRECT3DRMWRAP* *lplpD3DRMWrap*
);

Parameters

type
One of the members of the **D3DRMWRAPTYPE** enumerated type.

lpRef
Reference frame for the wrap.

ox, oy, and *oz*
Origin of the wrap.

dx, dy, and *dz*
The z-axis of the wrap.

ux, uy, and *uz*
The y-axis of the wrap.

ou and *ov*
Origin in the texture.

su and *sv*
Scale factor in the texture.

lplpD3DRMWrap
Address that will be filled with a pointer to an *IDirect3DRMWrap* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::EnumerateObjects

Calls the callback function specified by the *func* parameter on each of the active Direct3DRM objects.

HRESULT EnumerateObjects(
D3DRMOBJECTCALLBACK *func***,**
LPVOID *lpArg*
);

Parameters

func
Application-defined **D3DRMOBJECTCALLBACK** callback function to be called with each Direct3DRMObject object and the application-defined argument.

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

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::GetDevices

Returns all the Direct3DRM devices that have been created in the system.

HRESULT GetDevices(
LPDIRECT3DRMDEVICEARRAY* *lplpDevArray*
);

Parameters

lplpDevArray
Address of a pointer that will be filled with the resulting array of Direct3DRM devices. For information about the Direct3DRMDeviceArray object, see the *IDirect3DRMDeviceArray* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::GetNamedObject

Finds a Direct3DRMObject, given its name.

HRESULT GetNamedObject(
const char * *lpName***,**
LPDIRECT3DRMOBJECT* *lplpD3DRMObject*
);

Parameters

lpName
Name of the object to be searched for.

lplpD3DRMObject
Address of a pointer to be initialized with a valid Direct3DRMObject pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the system does not find an object with the name specified in the *lpName* parameter, this method returns D3DRM_OK but the *lplpD3DRMObject* parameter is NULL.

IDirect3DRM::GetSearchPath

Returns the current search path. For Windows, the path is a list of directories separated by semicolons (;).

HRESULT GetSearchPath(
DWORD * *lpdwSize***,**
LPSTR *lpszPath*
);

Parameters

lpdwSize
Pointer to a DWORD. Should contain the length of *lpszPath* when **GetSearchPath** is called. On return, contains the length of the string in *lpszPath* containing the current search path. This parameter cannot be NULL.

lpszPath
On return, contains a pointer to a null-terminated string specifying the search path. If *lpszPath* equals NULL when **GetSearchPath** is called, the method returns the length of the current string in the location pointed to by the *lpdwSize* parameter.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRM::SetSearchPath

IDirect3DRM::Load

Loads an object.

HRESULT Load(
LPVOID *lpvObjSource***,**
LPVOID *lpvObjID***,**
LPIID * *lplpGUIDs***,**
DWORD *dwcGUIDs***,**
D3DRMLOADOPTIONS *d3drmLOFlags***,**
D3DRMLOADCALLBACK *d3drmLoadProc***,**
LPVOID *lpArgLP***,**
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc***,**
LPVOID *lpArgLTP***,**
LPDIRECT3DRMFRAME *lpParentFrame*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

lplpGUIDs
Address of an array of interface identifiers to be loaded. For example, if this parameter is a two-element array containing IID_IDirect3DRMMeshBuilder and IID_IDirect3DRMAnimationSet, this method loads all the animation sets and mesh-builder objects. Possible GUIDs must be one or more of the following: IID_IDirect3DRMMeshBuilder, IID_IDirect3DRMAnimationSet, IID_IDirect3DRMAnimation, and IID_IDirect3DRMFrame.

dwcGUIDs
Number of elements specified in the *lplpGUIDs* parameter.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadProc
A **D3DRMLOADCALLBACK** callback function called when the system reads the specified object.

lpArgLP
Address of application-defined data passed to the **D3DRMLOADCALLBACK** callback function.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by an object that require special formatting. This parameter can be NULL.

lpArgLTP
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

lpParentFrame
Address of a parent Direct3DRMFrame object. This argument only affects the loading of animation sets. When an animation that is loaded from an X file references an unparented frame in the X file, its parent is set to this parent frame argument. However, if you ask **Load** to load any frames in the X file, the parent frame argument will not be used as the parent frame for frames in the X file with no parent. That is, the parent frame argument is used only when you load animation sets. This value of this argument can be NULL.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method allows great flexibility in loading objects from DirectX files. Here is an example of how to use it:

```
#include <windows.h>
#include <stdio.h>
#include <initguid.h>
#include <d3drm.h>

LPDIRECT3DRM d3dApi;

int totalMesh = 0, totalAnim = 0, totalFrames = 0;

HRESULT loadTexture(LPSTR name, LPVOID lpArg, LPDIRECT3DRMTEXTURE *tex)
{
    return d3dApi->lpVtbl->LoadTexture(d3dApi, name, tex);
}

void loadCallback(LPDIRECT3DRMOBJECT lpObject, REFIID objGuid, LPVOID
lpArg)
{
    if (IsEqualIID(objGuid, &IID_IDirect3DRMFrame)) {
        totalFrames ++;
        return;
    }

    if (IsEqualIID(objGuid, &IID_IDirect3DRMAnimationSet)) {
        totalAnim ++;
        return;
    }

    if (IsEqualIID(objGuid, &IID_IDirect3DRMMeshBuilder)) {
        totalMesh ++;
        return;
    }

    return;
}


BOOL loadObjects(LPSTR filename)
{
    LPGUID guids[] = { (LPGUID)&IID_IDirect3DRMMeshBuilder,
        (LPGUID)&IID_IDirect3DRMAnimationSet,
        (LPGUID)&IID_IDirect3DRMFrame };

    /* Tell the loader which objects you're interested in */

    if (FAILED(d3dApi->lpVtbl->Load(d3dApi, filename, NULL,
            guids, 3, D3DRMLOAD_FROMFILE,
```

```
            loadCallback, NULL, loadTexture, NULL, NULL)))
        return FALSE;

    printf("Total Frames loaded = %d\n", totalFrames);
    printf("Total Animation Sets loaded = %d\n", totalAnim);
    printf("Total Meshbuilders loaded = %d\n", totalMesh);

    return TRUE;
}

int main(int argc, char **argv)
{

    if (FAILED(Direct3DRMCreate(&d3dApi)))
        return FALSE;

    if (argc != 2) {
        fprintf(stderr, "usage: %s filename\n", argv[0]);
        return FALSE;
    }

    if (!loadObjects(argv[1])) return FALSE;

  return(0);
}
```

IDirect3DRM::LoadTexture

Loads a texture from the specified file. This texture can have 8, 24, or 32 bits-per-pixel, and it should be in either the Windows bitmap (.bmp) or Portable Pixmap (.ppm) P6 format.

HRESULT LoadTexture(
const char * *lpFileName*,
LPDIRECT3DRMTEXTURE* *lplpD3DRMTexture*
);

Parameters

lpFileName
Name of the required .bmp or .ppm file.

lplpD3DRMTexture
Address of a pointer to be initialized with a valid Direct3DRMTexture pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

LoadTexture checks whether the texture is in BMP or PPM format, which are the formats it knows how to load. If you want to load other formats, you can add code to the callback to load the image into a **D3DRMIMAGE** structure and then call **IDirect3DRM::CreateTexture**.

If you write your own texture callback, the **LoadTexture** call in the texture callback does not take a reference to the texture. For example:

```
HRESULT loadTextures(char *name, void *arg, LPDIRECT3DRMTEXTURE *tex)
{
    return lpD3DRM->LoadTexture(name, tex);
}
```

In this sample, no reference is taken for *tex*. If you want to keep a reference to each texture loaded by your texture callback, you should **AddRef** the texture. For example:

```
LPDIRECT3DRMTEXTURE *texarray;

HRESULT loadTextures(char *name, void *arg, LPDIRECT3DRMTEXTURE *tex)
{
    if (FAILED(lpD3DRM->LoadTexture(name, tex)) {
        return NULL;
    }

    texArray[current++] = tex;
    tex->AddRef();

    return tex;
}
```

IDirect3DRM::LoadTextureFromResource

Loads a texture from a specified resource.

HRESULT LoadTextureFromResource(
HRSRC *rs*,
LPDIRECT3DRMTEXTURE * *lplpD3DRMTexture*
);

Parameters

rs
Handle of the resource.

lplpD3DRMTexture
Address of a pointer to be initialized with a valid Direct3DRMTexture object if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM::SetDefaultTextureColors

Sets the default colors to be used for a Direct3DRMTexture object.

HRESULT SetDefaultTextureColors(
DWORD *dwColors*
);

Parameters *dwColors*
Number of colors.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method affects the texture colors only when it is called before the **IDirect3DRM::CreateTexture** method; it has no effect on textures that have already been created.

IDirect3DRM::SetDefaultTextureShades

Sets the default shades to be used for an Direct3DRMTexture object.

HRESULT SetDefaultTextureShades(
DWORD *dwShades*
);

Parameters *dwShades*
Number of shades.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method affects the texture shades only when it is called before the **IDirect3DRM::CreateTexture** method; it has no effect on textures that have already been created.

IDirect3DRM::SetSearchPath

Sets the current file search path from a list of directories. For Windows, the path should be a list of directories separated by semicolons (;).

HRESULT SetSearchPath(
LPCSTR *lpPath*
);

Parameters

lpPath
Address of a null-terminated string specifying the path to set as the current search path.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The default search path is taken from the value of the D3DPATH environment variable. If this is not set, the search path will be empty. When opening a file, the system first looks for the file in the current working directory and then checks every directory in the search path.

See Also

IDirect3DRM::GetSearchPath

IDirect3DRM::Tick

Performs the Direct3DRM system heartbeat. When this method is called, the positions of all moving frames are updated according to their current motion attributes, the scene is rendered to the current device, and relevant callback functions are called at their appropriate times. Control is returned when the rendering cycle is complete.

HRESULT Tick(
D3DVALUE *d3dvalTick*
);

Parameters

d3dvalTick
Velocity and rotation step for the **IDirect3DRMFrame::SetRotation** and **IDirect3DRMFrame::SetVelocity** methods.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks You can implement this method by using other Retained-Mode methods to allow more flexibility in rendering a scene.

IDirect3DRM2

Applications use the methods of the **IDirect3DRM2** interface to create Direct3DRM2 objects and work with system-level variables.

Applications use the methods of the **IDirect3DRMDevice2** and **IDirect3DRMDevice** interfaces to interact with the output device. While the **IDirect3DRMDevice** interface, when created from the **IDirect3DRM** interface, works with an **IDirect3DDevice** Immediate-Mode device, the **IDirect3DRMDevice2** interface, when created from the **IDirect3DRM2** interface or initialized by the **IDirect3DRMDevice2::InitFromClipper**, **IDirect3DRMDevice2::InitFromD3D2**, or **IDirect3DRMDevice2::InitFromSurface** method, works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRM2 supports all the methods in **IDirect3DRM**. In addition, the **IDirect3DRM2::CreateProgressiveMesh** method had been added. The **IDirect3DRM2::CreateDeviceFromSurface**, **IDirect3DRM2::CreateDeviceFromD3D**, and **IDirect3DRM2::CreateDeviceFromClipper** methods all create a DIRECT3DRMDEVICE2 object. The **IDirect3DRM2::CreateViewport** method creates a viewport on a DirectDRMDevice2 object. The **IDirect3DRM2::LoadTexture** and **IDirect3DRM2::LoadTextureFromResource** methods load a Direct3DRMTexture2 object.

To access the **IDirect3DRM2** COM interface, create an **IDirect3DRM** object with **Direct3DRMCreate**, then query for **IDirect3DRM2** from **IDirect3DRM**.

For a conceptual overview, see *IDirect3DRM and IDirect3DRM2 Interfaces*.

The methods of the **IDirect3DRM2** interface can be organized into the following groups:

Animation	**CreateAnimation**
	CreateAnimationSet
Devices	**CreateDevice**
	CreateDeviceFromClipper
	CreateDeviceFromD3D

	CreateDeviceFromSurface
	GetDevices
Enumeration	**EnumerateObjects**
Faces	**CreateFace**
Frames	**CreateFrame**
Lights	**CreateLight**
	CreateLightRGB
Materials	**CreateMaterial**
Meshes	**CreateMesh**
	CreateMeshBuilder
Miscellaneous	**CreateObject**
	CreateUserVisual
	GetNamedObject
	Load
	Tick
Progressive Meshes	**CreateProgressiveMesh**
Search paths	**AddSearchPath**
	GetSearchPath
	SetSearchPath
Shadows	**CreateShadow**
Textures	**CreateTexture**
	CreateTextureFromSurface
	LoadTexture
	LoadTextureFromResource
	SetDefaultTextureColors

	SetDefaultTextureShades
Viewports	**CreateViewport**
Wraps	**CreateWrap**

The **IDirect3DRM2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

IDirect3DRM2::AddSearchPath

Adds a list of directories to the end of the current file search path.

HRESULT AddSearchPath(
LPCSTR *lpPath*
);

Parameters

lpPath
Address of a null-terminated string specifying the path to add to the current search path.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

For Windows, the path should be a list of directories separated by semicolons (;).

See Also

IDirect3DRM2::SetSearchPath

IDirect3DRM2::CreateAnimation

Creates an empty Direct3DRMAnimation object.

HRESULT CreateAnimation(
LPDIRECT3DRMANIMATION * *lplpD3DRMAnimation*
);

Parameters *lplpD3DRMAnimation*
Address that will be filled with a pointer to an *IDirect3DRMAnimation* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateAnimationSet

Creates an empty Direct3DRMAnimationSet object.

HRESULT CreateAnimationSet (
LPDIRECT3DRMANIMATIONSET * *lplpD3DRMAnimationSet*
);

Parameters *lplpD3DRMAnimationSet*
Address that will be filled with a pointer to an *IDirect3DRMAnimationSet* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateDevice

Not implemented on the Windows platform.

HRESULT CreateDevice(
DWORD *dwWidth*,
DWORD *dwHeight*,
LPDIRECT3DRMDEVICE* *lplpD3DRMDevice*
);

IDirect3DRM2::CreateDeviceFromClipper

Creates a Direct3DRMDevice2 Windows device by using a specified DirectDrawClipper object. An **IDirect3DRMDevice2** interface works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device

supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

HRESULT CreateDeviceFromClipper(
LPDIRECTDRAWCLIPPER *lpDDClipper***,**
LPGUID *lpGUID***,**
int *width***,**
int *height***,**
LPDIRECT3DRMDEVICE2 * *lplpD3DRMDevice*
);

Parameters

lpDDClipper
Address of a DirectDrawClipper object.

lpGUID
Address of a globally unique identifier (GUID). This parameter can be NULL.

width and *height*
Width and height of the device to be created.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the *lpGUID* parameter is NULL, the system searches for a device with a default set of device capabilities. This is the recommended way to create a Retained-Mode device because it always works, even if the user installs new hardware.

The system describes the default settings by using the following flags from the **D3DPRIMCAPS** structure in internal device-enumeration calls:

D3DPSHADECAPS_ALPHAFLATSTIPPLED

D3DPTBLENDCAPS_COPY | **D3DPTBLENDCAPS_MODULATE**

D3DPMISCCAPS_CULLCCW

D3DPRASTERCAPS_FOGVERTEX

D3DPCMPCAPS_LESSEQUAL

D3DPTFILTERCAPS_NEAREST

D3DPTEXTURECAPS_PERSPECTIVE |
D3DPTEXTURECAPS_TRANSPARENCY

D3DPTADDRESSCAPS_WRAP

If a hardware device is not found, the monochromatic (ramp) software driver is loaded. An application should enumerate devices instead of specifying NULL in the *lpGUID* parameter if it has special needs that are not met by this list of default settings.

IDirect3DRM2::CreateDeviceFromD3D

Creates a Direct3DRMDevice2 Windows device by using specified Direct3D objects. An **IDirect3DRMDevice2** interface works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

HRESULT CreateDeviceFromD3D(
LPDIRECT3D2 *lpD3D*,
LPDIRECT3DDEVICE2 *lpD3DDev*,
LPDIRECT3DRMDEVICE2 * *lplpD3DRMDevice*
);

Parameters

lpD3D
Address of an instance of Direct3D2.

lpD3DDev
Address of a Direct3D2 device object.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateDeviceFromSurface

Creates a Direct3DRMDevice2 Windows device for rendering from the specified DirectDraw surfaces. An **IDirect3DRMDevice2** interface works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

HRESULT CreateDeviceFromSurface(
LPGUID *lpGUID*,
LPDIRECTDRAW *lpDD*,
LPDIRECTDRAWSURFACE *lpDDSBack*,
LPDIRECT3DRMDEVICE2 * *lplpD3DRMDevice*
);

Parameters

lpGUID
Address of the globally unique identifier (GUID) used as the required device driver. If this parameter is NULL, the default device driver is used.

lpDD
Address of the DirectDraw object that is the source of the DirectDraw surface.

lpDDSBack
Address of the DirectDrawSurface object that represents the back buffer.

lplpD3DRMDevice
Address that will be filled with a pointer to an *IDirect3DRMDevice2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateFace

Creates an instance of the *IDirect3DRMFace* interface.

HRESULT CreateFace(
LPDIRECT3DRMFACE * *lplpd3drmFace*
);

Parameters

lplpd3drmFace
Address that will be filled with a pointer to an *IDirect3DRMFace* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateFrame

Creates a new child frame (a Direct3DRMFrame2 object) of the given parent frame.

HRESULT CreateFrame(
LPDIRECT3DRMFRAME *lpD3DRMFrame*,
LPDIRECT3DRMFRAME2* *lplpD3DRMFrame2*
);

Parameters

lpD3DRMFrame
Address of a DIRECT3DRMFRAME object that is to be the parent of the new frame.

lplpD3DRMFrame
Address that will be filled with a pointer to an *IDirect3DRMFrame2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The child frame inherits the motion attributes of its parent. For example, if the parent is moving with a given velocity, the child frame will also move with that velocity. Furthermore, if the parent is set rotating, the child frame will rotate about the origin of the parent. Frames that have no parent are called scenes. To create a scene, specify NULL as the parent. An application can create a frame with no parent and then associate it with a parent frame later by using the **IDirect3DRMFrame2::AddChild** method.

See Also

IDirect3DRMFrame2::AddChild

IDirect3DRM2::CreateLight

Creates a new light source with the given type and color.

HRESULT CreateLight(
D3DRMLIGHTTYPE *d3drmltLightType***,**
D3DCOLOR *cColor***,**
LPDIRECT3DRMLIGHT* *lplpD3DRMLight*
);

Parameters

d3drmltLightType
One of the lighting types given in the **D3DRMLIGHTTYPE** enumerated type.

cColor
Color of the light.

lplpD3DRMLight
Address that will be filled with a pointer to an *IDirect3DRMLight* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateLightRGB

Creates a new light source with the given type and color.

HRESULT CreateLightRGB(
D3DRMLIGHTTYPE *ltLightType*,
D3DVALUE *vRed*,
D3DVALUE *vGreen*,
D3DVALUE *vBlue*,
LPDIRECT3DRMLIGHT* *lplpD3DRMLight*
);

Parameters

ltLightType
One of the lighting types given in the **D3DRMLIGHTTYPE** enumerated type.

vRed, vGreen, and *vBlue*
Color of the light.

lplpD3DRMLight
Address that will be filled with a pointer to an *IDirect3DRMLight* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateMaterial

Creates a material with the given specular property.

HRESULT CreateMaterial(
D3DVALUE *vPower*,
LPDIRECT3DRMMATERIAL * *lplpD3DRMMaterial*
);

Parameters

vPower
Sharpness of the reflected highlights, with a value of 5 giving a metallic look and higher values giving a more plastic look to the rendered surface.

lplpD3DRMMaterial
Address that will be filled with a pointer to an *IDirect3DRMMaterial* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateMesh

Creates a new mesh object with no faces. The mesh is not visible until it is added to a frame.

HRESULT CreateMesh(
LPDIRECT3DRMMESH* *lplpD3DRMMesh*
);

Parameters *lplpD3DRMMesh*
Address that will be filled with a pointer to an *IDirect3DRMMesh* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateMeshBuilder

Creates a new mesh builder object.

HRESULT CreateMeshBuilder(
LPDIRECT3DRMMESHBUILDER2* *lplpD3DRMMeshBuilder2*
);

Parameters *lplpD3DRMMeshBuilder*
Address that will be filled with a pointer to an *IDirect3DRMMeshBuilder2* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateObject

Creates a new object without initializing the object.

HRESULT CreateObject(
REFCLSID *rclsid*,
LPUNKNOWN *pUnkOuter*,
REFIID *riid*,
LPVOID FAR* *ppv*
);

Parameters

rclsid
Class identifier for the new object.

pUnkOuter
Allows COM aggregation features.

riid
Interface identifier of the object to be created.

ppv
Address of a pointer to the object when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An application that calls this method must initialize the object that has been created. (The other creation methods of the *IDirect3DRM2* interface initialize the object automatically.) To initialize the new object, you should use the **Init** method for that object. An application should call the **Init** method only once to initialize any given object.

Applications can use this method to implement aggregation in Direct3DRM objects.

IDirect3DRM2::CreateProgressiveMesh

Creates a new progressive mesh object with no faces. The mesh is not visible until it is added to a frame.

HRESULT CreateProgressiveMesh(
LPDIRECT3DRMPROGRESSIVEMESH* *lplpD3DRMProgressiveMesh*
);

Parameters

lplpD3DRMProgressiveMesh
Address that will be filled with a pointer to an *IDirect3DRMProgressiveMesh* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateShadow

Creates a shadow by using the specified visual and light, projecting the shadow onto the specified plane. The shadow is a visual that should be added to the frame that contains the visual.

HRESULT CreateShadow(
LPDIRECT3DRMVISUAL *lpVisual*,
LPDIRECT3DRMLIGHT *lpLight*,
D3DVALUE *px*,
D3DVALUE *py*,
D3DVALUE *pz*,
D3DVALUE *nx*,
D3DVALUE *ny*,
D3DVALUE *nz*,
LPDIRECT3DRMVISUAL * *lplpShadow*
);

Parameters

lpVisual
Address of the Direct3DRMVisual object that is casting the shadow.

lpLight
Address of the *IDirect3DRMLight* interface that is the light source.

px, py, and *pz*
Plane that the shadow is to be projected on.

nx, ny, and *nz*
Normal to the plane that the shadow is to be projected on.

lplpShadow
Address of a pointer to be initialized with a valid pointer to the shadow visual, if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateTexture

Creates a texture from an image in memory.

HRESULT CreateTexture(
LPD3DRMIMAGE *lpImage*,
LPDIRECT3DRMTEXTURE2* *lplpD3DRMTexture2*

);

Parameters

lpImage
Address of a **D3DRMIMAGE** structure describing the source for the texture.

lplpD3DRMTexture2
Address that will be filled with a pointer to an *IDirect3DRMTexture2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The memory associated with the image is used each time the texture is rendered, rather than the memory being copied into Direct3DRM's buffers. This allows the image to be used both as a rendering target and as a texture.

IDirect3DRM2::CreateTextureFromSurface

Creates a texture from a specified DirectDraw surface.

HRESULT CreateTextureFromSurface(
LPDIRECTDRAWSURFACE *lpDDS*,
LPDIRECT3DRMTEXTURE2 * *lplpD3DRMTexture2*
);

Parameters

lpDDS
Address of the DirectDrawSurface object containing the texture.

lplpD3DRMTexture
Address that will be filled with a pointer to an *IDirect3DRMTexture2* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateUserVisual

Creates an application-defined visual object, which can then be added to a scene and rendered by using an application-defined handler.

HRESULT CreateUserVisual(
D3DRMUSERVISUALCALLBACK *fn*,

LPVOID *lpArg*,
LPDIRECT3DRMUSERVISUAL * *lplpD3DRMUV*
);

Parameters

fn
Application-defined **D3DRMUSERVISUALCALLBACK** callback function.

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

lplpD3DRMUV
Address that will be filled with a pointer to an *IDirect3DRMUserVisual* interface if the call succeeds.

Return Values

Should return D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::CreateViewport

Creates a viewport on a Direct3DRMDevice2 device with device coordinates (*dwXPos*, *dwYPos*) to (*dwXPos* + *dwWidth*, *dwYPos* + *dwHeight*).

HRESULT CreateViewport(
LPDIRECT3DRMDEVICE2 *lpDev*,
LPDIRECT3DRMFRAME *lpCamera*,
DWORD *dwXPos*,
DWORD *dwYPos*,
DWORD *dwWidth*,
DWORD *dwHeight*,
LPDIRECT3DRMVIEWPORT* *lplpD3DRMViewport*
);

Parameters

lpDev
Address of a Direct3DRMDevice2 device on which the viewport is to be created.

lpCamera
Address of a frame that describes the position and direction of the view.

dwXPos, *dwYPos*, *dwWidth*, and *dwHeight*
Position and size of the viewport, in device coordinates. The viewport size cannot be larger than the physical device or the method will fail.

lplpD3DRMViewport
Address that will be filled with a pointer to an *IDirect3DRMViewport* interface if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. If the viewport size is larger than the physical device, returns D3DRMERR_BADVALUE. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The viewport displays objects in the scene that contains the camera, with the view direction and up vector taken from the camera. The viewport size cannot be larger than the physical device.

An **IDirect3DRMDevice2** interface works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRM2::CreateWrap

Creates a wrapping function that can be used to assign texture coordinates to faces and meshes. The vector [*ox oy oz*] gives the origin of the wrap, [*dx dy dz*] gives its z-axis, and [*ux uy uz*] gives its y-axis. The 2D vectors [*ou ov*] and [*su sv*] give an origin and scale factor in the texture applied to the result of the wrapping function.

HRESULT CreateWrap(
D3DRMWRAPTYPE *type*,
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *ox*,
D3DVALUE *oy*,
D3DVALUE *oz*,
D3DVALUE *dx*,
D3DVALUE *dy*,
D3DVALUE *dz*,
D3DVALUE *ux*,
D3DVALUE *uy*,
D3DVALUE *uz*,
D3DVALUE *ou*,
D3DVALUE *ov*,
D3DVALUE *su*,
D3DVALUE *sv*,
LPDIRECT3DRMWRAP* *lplpD3DRMWrap*
);

Parameters

type
One of the members of the **D3DRMWRAPTYPE** enumerated type.

lpRef
Reference frame for the wrap.

ox, oy, and *oz*
Origin of the wrap.

dx, dy, and *dz*
The z-axis of the wrap.

ux, uy, and *uz*
The y-axis of the wrap.

ou and *ov*
Origin in the texture.

su and *sv*
Scale factor in the texture.

lplpD3DRMWrap
Address that will be filled with a pointer to an *IDirect3DRMWrap* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::EnumerateObjects

Calls the callback function specified by the *func* parameter on each of the active Direct3DRM objects.

HRESULT EnumerateObjects(
D3DRMOBJECTCALLBACK *func*,
LPVOID *lpArg*
);

Parameters *func*
Application-defined **D3DRMOBJECTCALLBACK** callback function to be called with each Direct3DRMObject object and the application-defined argument.

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

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::GetDevices

Returns all the Direct3DRM devices that have been created in the system.

HRESULT GetDevices(
LPDIRECT3DRMDEVICEARRAY* *lplpDevArray*
);

Parameters

lplpDevArray
Address of a pointer that will be filled with the resulting array of Direct3DRM devices. For information about the Direct3DRMDeviceArray object, see the *IDirect3DRMDeviceArray* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::GetNamedObject

Finds a Direct3DRMObject, given its name.

HRESULT GetNamedObject(
const char * *lpName***,**
LPDIRECT3DRMOBJECT* *lplpD3DRMObject*
);

Parameters

lpName
Name of the object to be searched for.

lplpD3DRMObject
Address of a pointer to be initialized with a valid Direct3DRMObject pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the system does not find an object with the name specified in the *lpName* parameter, this method returns D3DRM_OK but the *lplpD3DRMObject* parameter is NULL.

IDirect3DRM2::GetSearchPath

Returns the current search path. For Windows, the path is a list of directories separated by semicolons (;).

HRESULT GetSearchPath(
DWORD * *lpdwSize***,**
LPSTR *lpszPath*
);

Parameters

lpdwSize
Pointer to a DWORD. Should contain the length of *lpszPath* when **GetSearchPath** is called. On return, contains the length of the string in *lpszPath* containing the current search path. This parameter cannot be NULL.

lpszPath
On return, contains a pointer to a null-terminated string specifying the search path. If *lpszPath* equals NULL when **GetSearchPath** is called, the method returns the length of the current string in the location pointed to by the *lpdwSize* parameter.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRM2::SetSearchPath

IDirect3DRM2::Load

Loads an object.

HRESULT Load(
LPVOID *lpvObjSource***,**
LPVOID *lpvObjID***,**
LPIID * *lplpGUIDs***,**
DWORD *dwcGUIDs***,**
D3DRMLOADOPTIONS *d3drmLOFlags***,**
D3DRMLOADCALLBACK *d3drmLoadProc***,**
LPVOID *lpArgLP***,**
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc***,**
LPVOID *lpArgLTP***,**
LPDIRECT3DRMFRAME *lpParentFrame*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

lplpGUIDs
Address of an array of interface identifiers to be loaded. For example, if this parameter is a two-element array containing IID_IDirect3DRMMeshBuilder and IID_IDirect3DRMAnimationSet, this method loads all the animation sets and mesh-builder objects. Possible GUIDs must be one or more of the following: IID_IDirect3DRMMeshBuilder, IID_IDirect3DRMAnimationSet, IID_IDirect3DRMAnimation, and IID_IDirect3DRMFrame.

dwcGUIDs
Number of elements specified in the *lplpGUIDs* parameter.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadProc
A **D3DRMLOADCALLBACK** callback function called when the system reads the specified object.

lpArgLP
Address of application-defined data passed to the **D3DRMLOADCALLBACK** callback function.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by an object that require special formatting. This parameter can be NULL.

lpArgLTP
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

lpParentFrame
Address of a parent Direct3DRMFrame object. This argument only affects the loading of animation sets. When an animation that is loaded from an X file references an unparented frame in the X file, its parent is set to this parent frame argument. However, if you ask **Load** to load any frames in the X file, the parent frame argument will not be used as the parent frame for frames in the X file with no parent. That is, the parent frame argument is used only when you load animation sets. This value of this argument can be NULL.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method allows great flexibility in loading objects from DirectX files. See **IDirect3DRM::Load** for an example of how to use it.

IDirect3DRM2::LoadTexture

Loads a Direct3DRMTexture2 texture from the specified file. This texture can have 8, 24, or 32 bits-per-pixel, and it should be in either the Windows bitmap (.bmp) or Portable Pixmap (.ppm) P6 format.

HRESULT LoadTexture(
const char * *lpFileName***,**
LPDIRECT3DRMTEXTURE2* *lplpD3DRMTexture*
);

Parameters

lpFileName
Address of the name of the required .bmp or .ppm file.

lplpD3DRMTexture
Address of a pointer to be initialized with a valid **IDirect3DRMTexture2** pointer if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks **LoadTexture** checks whether the texture is in BMP or PPM format, which are the formats it knows how to load. If you want to load other formats, you can add code to the callback to load the image into a **D3DRMIMAGE** structure and then call **IDirect3DRM2::CreateTexture**.

If you write your own texture callback, the **LoadTexture** call in the texture callback does not take a reference to the texture. For example:

```
HRESULT loadTextures(char *name, void *arg, LPDIRECT3DRMTEXTURE2 *tex)
{
    return lpD3DRM2->LoadTexture(name, tex);
}
```

In this sample, no reference is taken for *tex*. If you want to keep a reference to each texture loaded by your texture callback, you should **AddRef** the texture. For example:

```
LPDIRECT3DRMTEXTURE2 *texarray;

HRESULT loadTextures(char *name, void *arg, LPDIRECT3DRMTEXTURE2 *tex)
{
    if (FAILED(lpD3DRM2->LoadTexture(name, tex)) {
```

```
        return NULL;
    }

    texArray[current++] = tex;
    tex->AddRef();

    return tex;
}
```

IDirect3DRM2::LoadTextureFromResource

Loads a Direct3DRMTexture2 texture from a specified resource.

HRESULT LoadTextureFromResource(
HMODULE *hModule*,
LPCTSTR *strName*,
LPCTSTR *strType*,
LPDIRECT3DRMTEXTURE2 * *lplpD3DRMTexture*
);

Parameters

hModule
Handle of the module whose executable file contains the resource.

strName
A null-terminated string specifying the name of the resource to be used as the texture.

strType
A null-terminated string specifying the type name of the resource. This parameter can be one of the following resource types or a user-defined type:

Value	Meaning
RT_BITMAP	Bitmap reource
RT_RCDATA	Application-defined resource (raw data)

lplpD3DRMTexture
Address of a pointer to be initialized with a valid **IDirect3DRMTexture2** object if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRM2::SetDefaultTextureColors

Sets the default colors to be used for a Direct3DRMTexture2 object.

HRESULT SetDefaultTextureColors(
DWORD *dwColors*
);

Parameters

dwColors
Number of colors.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method affects the texture colors only when it is called before the **IDirect3DRM2::CreateTexture** method; it has no effect on textures that have already been created.

IDirect3DRM2::SetDefaultTextureShades

Sets the default shades to be used for an Direct3DRMTexture2 object.

HRESULT SetDefaultTextureShades(
DWORD *dwShades*
);

Parameters

dwShades
Number of shades.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method affects the texture shades only when it is called before the **IDirect3DRM2::CreateTexture** method; it has no effect on textures that have already been created.

IDirect3DRM2::SetSearchPath

Sets the current file search path from a list of directories. For Windows, the path should be a list of directories separated by semicolons (;).

HRESULT SetSearchPath(
LPCSTR *lpPath*
);

Parameters

lpPath
Address of a null-terminated string specifying the path to set as the current search path.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The default search path is taken from the value of the D3DPATH environment variable. If this is not set, the search path will be empty. When opening a file, the system first looks for the file in the current working directory and then checks every directory in the search path.

D3DPATH is an environment variable that sets the default search path.

See Also

IDirect3DRM2::GetSearchPath

IDirect3DRM2::Tick

Performs the Direct3DRM system heartbeat. When this method is called, the positions of all moving frames are updated according to their current motion attributes, the scene is rendered to the current device, and relevant callback functions are called at their appropriate times. Control is returned when the rendering cycle is complete.

HRESULT Tick(
D3DVALUE *d3dvalTick*
);

Parameters

d3dvalTick
Velocity and rotation step for the **IDirect3DRMFrame::SetRotation** and **IDirect3DRMFrame::SetVelocity** methods.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks You can implement this method by using other Retained-Mode methods to allow more flexibility in rendering a scene.

IDirect3DRMAnimation

Applications use the methods of the **IDirect3DRMAnimation** interface to animate the position, orientation, and scaling of visuals, lights, and viewports. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMAnimation and IDirect3DRMAnimationSet Interfaces*.

The methods of the **IDirect3DRMAnimation** interface can be organized into the following groups:

Keys	**AddPositionKey**
	AddRotateKey
	AddScaleKey
	DeleteKey
Miscellaneous	**SetFrame**
	SetTime
Options	**GetOptions**
	SetOptions

The **IDirect3DRMAnimation** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMAnimation** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMAnimation object is obtained by calling the **IDirect3DRM::CreateAnimation** method.

IDirect3DRMAnimation::AddPositionKey

Adds a position key to the animation.

HRESULT AddPositionKey(
D3DVALUE *rvTime*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

rvTime
Time in the animation to store the position key. The time units are arbitrary and zero-based; a key whose *rvTime* value is 49 occurs exactly in the middle of an animation whose last key has an *rvTime* value of 99.

rvX, rvY, and *rvZ*
Position.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The transformation applied by this method is a translation. For information about the matrix mathematics involved in transformations, see **3D Transformations**.

See Also

IDirect3DRMAnimation::DeleteKey

IDirect3DRMAnimation::AddRotateKey

Adds a rotate key to the animation.

HRESULT AddRotateKey(
D3DVALUE *rvTime*,

D3DRMQUATERNION **rqQuat*
);

Parameters

rvTime
Time in the animation to store the rotate key. The time units are arbitrary and zero-based; a key whose *rvTime* value is 49 occurs exactly in the middle of an animation whose last key has an *rvTime* value of 99.

rqQuat
Quaternion representing the rotation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method applies a rotation transformation. For information about the matrix mathematics involved in transformations, see **3D Transformations**.

See Also

IDirect3DRMAnimation::DeleteKey

IDirect3DRMAnimation::AddScaleKey

Adds a scale key to the animation.

HRESULT AddScaleKey(
D3DVALUE *rvTime*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

rvTime
Time in the animation to store the scale key. The time units are arbitrary and zero-based; a key whose *rvTime* value is 49 occurs exactly in the middle of an animation whose last key has an *rvTime* value of 99.

rvX, rvY, and *rvZ*
Scale factor.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method applies a scaling transformation. For information about the matrix mathematics involved in transformations, see **3D Transformations**.

See Also

IDirect3DRMAnimation::DeleteKey

IDirect3DRMAnimation::DeleteKey

Removes a key from an animation.

HRESULT DeleteKey(
D3DVALUE *rvTime*
);

Parameters

rvTime
Time identifying the key that will be removed from the animation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMAnimation::GetOptions

Retrieves animation options.

D3DRMANIMATIONOPTIONS GetOptions( );

Return Values

Returns the value of the **D3DRMANIMATIONOPTIONS** type describing the animation options.

See Also

IDirect3DRMAnimation::SetOptions

IDirect3DRMAnimation::SetFrame

Sets the frame for the animation.

HRESULT SetFrame(
LPDIRECT3DRMFRAME *lpD3DRMFrame*
);

Parameters

lpD3DRMFrame
Address of a variable representing the frame to set for the animation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMAnimation::SetOptions

Sets the animation options.

HRESULT SetOptions(
D3DRMANIMATIONOPTIONS *d3drmanimFlags*
);

Parameters

d3drmanimFlags
Value of the **D3DRMANIMATIONOPTIONS** type describing the animation options.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMAnimation::GetOptions

IDirect3DRMAnimation::SetTime

Sets the current time for this animation.

HRESULT SetTime(
D3DVALUE *rvTime*
);

Parameters

rvTime
New current time for the animation. The time units are arbitrary and zero-based; a key whose *rvTime* value is 49 occurs exactly in the middle of an animation whose last key has an *rvTime* value of 99.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMAnimationSet

Applications use the methods of the **IDirect3DRMAnimationSet** interface to group Direct3DRMAnimation objects together, which can simplify the playback of complex animation sequences. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMAnimation and IDirect3DRMAnimationSet Interfaces*.

The methods of the **IDirect3DRMAnimationSet** interface can be organized into the following groups:

Adding, loading, and removing	**AddAnimation**
	DeleteAnimation
	Load
Time	**SetTime**

The **IDirect3DRMAnimationSet** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMAnimationSet** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMAnimationSet object is obtained by calling the **IDirect3DRM::CreateAnimationSet** method.

IDirect3DRMAnimationSet::AddAnimation

Adds an animation to the animation set.

HRESULT AddAnimation(
  LPDIRECT3DRMANIMATION *lpD3DRMAnimation*
  );

Parameters *lpD3DRMAnimation*
Address of the Direct3DRMAnimation object to be added to the animation set.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMAnimationSet::DeleteAnimation

Removes a previously added animation from the animation set.

HRESULT DeleteAnimation(
LPDIRECT3DRMANIMATION *lpD3DRMAnimation*
);

Parameters *lpD3DRMAnimation*
Address of the Direct3DRMAnimation object to be removed from the animation set.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMAnimationSet::Load

Loads an animation set.

HRESULT Load(
LPVOID *lpvObjSource*,
LPVOID *lpvObjID*,
D3DRMLOADOPTIONS *d3drmLOFlags*,
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc*,
LPVOID *lpArgLTP*,
LPDIRECT3DRMFRAME *lpParentFrame*
);

Parameters *lpvObjSource*
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by the object that require special formatting. This parameter can be NULL.

lpArgLTP
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

lpParentFrame
Address of a parent Direct3DRMFrame object. This argument only affects the loading of animation sets. When an animation that is loaded from an X file references an unparented frame in the X file, its parent is set to this parent frame argument. However, if you ask **Load** to load any frames in the X file, the parent frame argument will not be used as the parent frame for frames in the X file with no parent. That is, the parent frame argument is used only when you load animation sets. This value of this argument can be NULL.

Return Values
Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks
By default, this method loads the first animation set in the file specified by the *lpvObjSource* parameter.

IDirect3DRMAnimationSet::SetTime

Sets the time for this animation set.

HRESULT SetTime(
D3DVALUE *rvTime*
);

Parameters
rvTime
New time.

Return Values
Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice

Applications use the methods of the **IDirect3DRMDevice** interface to interact with the output device. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMDevice, IDirect3DRMDevice2, and IDirect3DRMDeviceArray Interfaces*.

The methods of the **IDirect3DRMDevice** interface can be organized into the following groups:

Buffer counts	**GetBufferCount**
	SetBufferCount
Color models	**GetColorModel**
Dithering	**GetDither**
	SetDither
Initialization	**Init**
	InitFromClipper
	InitFromD3D
Miscellaneous	**GetDirect3DDevice**
	GetHeight
	GetTrianglesDrawn
	GetViewports
	GetWidth
	GetWireframeOptions
	Update
Notifications	**AddUpdateCallback**
	DeleteUpdateCallback
Rendering quality	**GetQuality**
	SetQuality
Shading	**GetShades**
	SetShades

Texture quality	**GetTextureQuality**
	SetTextureQuality

The **IDirect3DRMDevice** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMDevice** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMDevice object is obtained by calling the **IDirect3DRM::CreateDevice** method.

IDirect3DRMDevice::AddUpdateCallback

Adds a callback function that alerts the application when a change occurs to the device. The system calls this callback function whenever the application calls the **IDirect3DRMDevice::Update** method.

HRESULT AddUpdateCallback(
D3DRMUPDATECALLBACK *d3drmUpdateProc*,
LPVOID *arg*
);

Parameters

d3drmUpdateProc
Address of an application-defined callback function, **D3DRMUPDATECALLBACK**.

arg
Private data to be passed to the update callback function.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice::DeleteUpdateCallback**, **IDirect3DRMDevice::Update**, **D3DRMUPDATECALLBACK**

IDirect3DRMDevice::DeleteUpdateCallback

Removes an update callback function that was added by calling the **IDirect3DRMDevice::AddUpdateCallback** method.

```
HRESULT DeleteUpdateCallback(
    D3DRMUPDATECALLBACK d3drmUpdateProc,
    LPVOID arg
    );
```

Parameters

d3drmUpdateProc
Address of an application-defined callback function, **D3DRMUPDATECALLBACK**.

arg
Private data that was passed to the update callback function.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice::AddUpdateCallback**, **IDirect3DRMDevice::Update**, **D3DRMUPDATECALLBACK**

IDirect3DRMDevice::GetBufferCount

Retrieves the value set in a call to the **IDirect3DRMDevice::SetBufferCount** method.

```
DWORD GetBufferCount( );
```

Return Values Returns the number of buffers—one for single-buffering, two for double-buffering, and so on.

IDirect3DRMDevice::GetColorModel

Retrieves the color model of a device.

D3DCOLORMODEL GetColorModel( );

Return Values Returns a value from the **D3DCOLORMODEL** enumerated type that describes the Direct3D color model (RGB or monochrome).

IDirect3DRMDevice::GetDirect3DDevice

Retrieves a pointer to an Immediate-Mode device.

HRESULT GetDirect3DDevice(
LPDIRECT3DDEVICE * *lplpD3DDevice*
);

Parameters lplpD3DDevice
Address of a pointer that is initialized with a pointer to an Immediate-Mode device object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice::GetDither

Retrieves the dither flag for the device.

BOOL GetDither( );

Return Values Returns TRUE if the dither flag is set, or FALSE otherwise.

See Also **IDirect3DRMDevice::SetDither**

IDirect3DRMDevice::GetHeight

Retrieves the height, in pixels, of a device. This method is a convenience function.

DWORD GetHeight( );

Return Values Returns the height.

IDirect3DRMDevice::GetQuality

Retrieves the rendering quality for the device.

D3DRMRENDERQUALITY GetQuality( );

Return Values Returns one or more of the members of the enumerated types represented by the **D3DRMRENDERQUALITY** type.

See Also **IDirect3DRMDevice::SetQuality**

IDirect3DRMDevice::GetShades

Retrieves the number of shades in a ramp of colors used for shading.

DWORD GetShades( );

Return Values Returns the number of shades.

See Also **IDirect3DRMDevice::SetShades**

IDirect3DRMDevice::GetTextureQuality

Retrieves the current texture quality parameter for the device. Texture quality is relevant only for an RGB device.

D3DRMTEXTUREQUALITY GetTextureQuality( );

Return Values Returns one of the members of the **D3DRMTEXTUREQUALITY** enumerated type.

See Also **IDirect3DRMDevice::SetTextureQuality**

IDirect3DRMDevice::GetTrianglesDrawn

Retrieves the number of triangles drawn to a device since its creation. This method is a convenience function.

DWORD GetTrianglesDrawn( );

Return Values Returns the number of triangles.

Remarks The number of triangles includes those that were passed to the renderer but were not drawn because they were backfacing. The number does not include triangles that were rejected for lying outside of the viewing frustum.

IDirect3DRMDevice::GetViewports

Constructs a Direct3DRMViewportArray object that represents the viewports currently constructed from the device.

HRESULT GetViewports(
LPDIRECT3DRMVIEWPORTARRAY* *lplpViewports*
);

Parameters *lplpViewports*
Address of a pointer that is initialized with a valid Direct3DRMViewportArray object if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice::GetWidth

Retrieves the width, in pixels, of a device. This method is a convenience function.

DWORD GetWidth( );

Return Values Returns the width.

IDirect3DRMDevice::GetWireframeOptions

Retrieves the wireframe options of a given device.

DWORD GetWireframeOptions( );

Return Values Returns a bitwise **OR** of the following values:

D3DRMWIREFRAME_CULL

The backfacing faces are not drawn.

D3DRMWIREFRAME_HIDDENLINE

Wireframe-rendered lines are obscured by nearer objects.

IDirect3DRMDevice::Init

Not implemented on the Windows platform.

HRESULT Init(
ULONG *width*,
ULONG *height*
);

IDirect3DRMDevice::InitFromClipper

Initializes a device from a specified DirectDrawClipper object.

HRESULT InitFromClipper(
LPDIRECTDRAWCLIPPER *lpDDClipper*,
LPGUID *lpGUID*,
int *width*,
int *height*
);

Parameters

lpDDClipper
Address of the DirectDrawClipper object to use as an initializer.

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

width and *height*
Width and height of the device.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice::InitFromD3D

Initializes a Retained-Mode device from a specified Direct3D Immediate-Mode object and Immediate-Mode device.

HRESULT InitFromD3D(
LPDIRECT3D *lpD3D*,
LPDIRECT3DDEVICE *lpD3DIMDev*
);

Parameters

lpD3D
Address of the Direct3D Immediate-Mode object to use to initialize the Retained-Mode device.

lpD3DIMDev
Address of the Immediate-Mode device to use to initialize the Retained-Mode device.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice::SetBufferCount

Sets the number of buffers currently being used by the application.

HRESULT SetBufferCount(
DWORD *dwCount*
);

Parameters

dwCount
Specifies the number of buffers—one for single-buffering, two for double-buffering, and so on. The default value is 1, which is correct only for single-buffered window operation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An application that employs double-buffering or triple-buffering must use this method to inform the system of how many buffers it is using so that the system can calculate how much of the window to clear and update on each frame.

See Also

IDirect3DRMDevice::GetBufferCount

IDirect3DRMDevice::SetDither

Sets the dither flag for the device.

HRESULT SetDither(
BOOL *bDither*
);

Parameters

bDither
New dithering mode for the device. The default is TRUE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMDevice::GetDither

IDirect3DRMDevice::SetQuality

Sets the rendering quality of a device

HRESULT SetQuality (
D3DRMRENDERQUALITY *rqQuality*
);

Parameters

rqQuality
One or more of the members of the enumerated types represented by the **D3DRMRENDERQUALITY** type. The default setting is D3DRMRENDER_FLAT.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The rendering quality is the maximum quality at which rendering can take place on the rendering surface of that device.

An object's quality has three components: shade mode (flat or gouraud, phong is not yet implemented and will default to gouraud shading), lighting type (on or off), and fill mode (point, wireframe or solid).

You set the quality of a device with **SetQuality**. By default it is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMProgressiveMesh, Direct3DRMMeshBuilder, or Direct3DRMMeshBuilder2 object with their respective **SetQuality** methods. By default, the quality of these objects is D3DRMRENDER_GOURAUD (gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects

ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

See Also **IDirect3DRMDevice::GetQuality**

IDirect3DRMDevice::SetShades

Sets the number of shades in a ramp of colors used for shading.

HRESULT SetShades(
DWORD *ulShades*
);

Parameters *ulShades*
New number of shades. This parameter must be a power of 2. The default is 32.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice::GetShades**

IDirect3DRMDevice::SetTextureQuality

Sets the texture quality for the device.

HRESULT SetTextureQuality(
D3DRMTEXTUREQUALITY *tqTextureQuality*
);

Parameters *tqTextureQuality*
One of the members of the **D3DRMTEXTUREQUALITY** enumerated type. The default is **D3DRMTEXTURE_NEAREST**.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice::GetTextureQuality**

IDirect3DRMDevice::Update

Copies the image that has been rendered to the display. It also provides a heartbeat function to the device driver.

HRESULT Update( );

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks Each call to this method causes the system to call an application-defined callback function, **D3DRMUPDATECALLBACK**. To add a callback function, use the **IDirect3DRMDevice::AddUpdateCallback** method.

See Also **IDirect3DRMDevice::AddUpdateCallback**, **D3DRMUPDATECALLBACK**

IDirect3DRMDevice2

Applications use the methods of the *IDirect3DRMDevice2* and **IDirect3DRMDevice** interfaces to interact with the output device. While **IDirect3DRMDevice**, when created from the **IDirect3DRM** interface, works with an **IDirect3DDevice** Immediate-Mode device, **IDirect3DRMDevice2**, when created from the **IDirect3DRM2** interface, or initialized by the **IDirect3DRMDevice2::InitFromD3D2** or **IDirect3DRMDevice2::InitFromSurface** method, works with an **IDirect3DDevice2** Immediate-Mode device. The **IDirect3DDevice2** device supports the **DrawPrimitive** interface as well as execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

The **IDirect3DRMDevice2::InitFromSurface** method uses the **IDirect3DRM2::CreateDevice** method to create an *IDirect3DRMDevice2* interface. The **IDirect3DRMDevice2::InitFromD3D2** method uses an **IDirect3D2** Immediate-Mode object and an **IDirect3DDevice2** Immediate-Mode device to initialize an **IDirect3DRMDevice2** Retained-Mode device.

You can still query back and forth between **IDirect3DRMDevice** and **IDirect3DRMDevice2**. The main difference is in how the underlying Immediate-Mode device is created.

IDirect3DRMDevice2 contains all the methods in **IDirect3DRMDevice** plus two addition ones that allow you to control transparency, **IDirect3DRMDevice2::GetRenderMode** and **IDirect3DRMDevice2::SetRenderMode**, one additional initialization method, **IDirect3DRMDevice2::InitFromSurface**, and two changed methods

IDirect3DRMDevice2::GetDirect3DDevice2 and **IDirect3DRMDevice2::InitFromD3D2** which get and initialize an **IDirect3DRMDevice2** object rather than an **IDirect3DRMDevice** object.

This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMDevice, IDirect3DRMDevice2, and IDirect3DRMDeviceArray Interfaces*.

The methods of the **IDirect3DRMDevice2** interface can be organized into the following groups:

Buffer counts	**GetBufferCount**
	SetBufferCount
Color models	**GetColorModel**
Dithering	**GetDither**
	SetDither
Initialization	**Init**
	InitFromClipper
	InitFromD3D2
	InitFromSurface
Miscellaneous	**GetDirect3DDevice2**
	GetHeight
	GetTrianglesDrawn
	GetViewports
	GetWidth
	GetWireframeOptions
	Update
Notifications	**AddUpdateCallback**
	DeleteUpdateCallback
Rendering quality	**GetQuality**
	SetQuality
Shading	**GetShades**

	SetShades
Texture quality	**GetTextureQuality** **SetTextureQuality**
Transparency	**GetRenderMode** **SetRenderMode**

The **IDirect3DRMDevice2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The **IUnknown** interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMDevice2** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMDevice2 object is obtained by calling the **IDirect3DRM2::CreateDevice** method.

IDirect3DRMDevice2::AddUpdateCallback

Adds a callback function that alerts the application when a change occurs to the device. The system calls this callback function whenever the application calls the **IDirect3DRMDevice2::Update** method.

HRESULT AddUpdateCallback(
D3DRMUPDATECALLBACK *d3drmUpdateProc*,
LPVOID *arg*
);

Parameters

d3drmUpdateProc
Address of an application-defined callback function, **D3DRMUPDATECALLBACK**.

arg
Private data to be passed to the update callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMDevice2::DeleteUpdateCallback, **IDirect3DRMDevice2::Update**, **D3DRMUPDATECALLBACK**

IDirect3DRMDevice2::DeleteUpdateCallback

Removes an update callback function that was added by calling the **IDirect3DRMDevice2::AddUpdateCallback** method.

HRESULT DeleteUpdateCallback(
D3DRMUPDATECALLBACK *d3drmUpdateProc*,
LPVOID *arg*
);

Parameters

d3drmUpdateProc
Address of an application-defined callback function, **D3DRMUPDATECALLBACK**.

arg
Private data that was passed to the update callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMDevice2::AddUpdateCallback, **IDirect3DRMDevice2::Update**, **D3DRMUPDATECALLBACK**

IDirect3DRMDevice2::GetBufferCount

Retrieves the value set in a call to the **IDirect3DRMDevice2::SetBufferCount** method.

DWORD GetBufferCount( );

Return Values Returns the number of buffers—one for single-buffering, two for double-buffering, and so on.

IDirect3DRMDevice2::GetColorModel

Retrieves the color model of a device.

D3DCOLORMODEL GetColorModel( );

Return Values Returns a value from the **D3DCOLORMODEL** enumerated type that describes the Direct3D color model (RGB or monochrome).

IDirect3DRMDevice2::GetDirect3DDevice2

Retrieves a pointer to an **IDirect3DDevice2** Immediate-Mode device.

HRESULT GetDirect3DDevice2(
LPDIRECT3DDEVICE2 * *lplpD3DDevice*
);

Parameters lplpD3DDevice
Address of a pointer that is initialized with a pointer to an **IDirect3DDevice2** Immediate-Mode device object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The **IDirect3DDevice2** device will support the DrawPrimitive interface and execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRMDevice2::GetDither

Retrieves the dither flag for the device.

BOOL GetDither( );

Return Values Returns TRUE if the dither flag is set, or FALSE otherwise.

See Also **IDirect3DRMDevice2::SetDither**

IDirect3DRMDevice2::GetHeight

Retrieves the height, in pixels, of a device. This method is a convenience function.

DWORD GetHeight( );

Return Values Returns the height.

IDirect3DRMDevice2::GetQuality

Retrieves the rendering quality for the device.

D3DRMRENDERQUALITY GetQuality( );

Return Values Returns one or more of the members of the enumerated types represented by the **D3DRMRENDERQUALITY** type.

See Also **IDirect3DRMDevice2::SetQuality**

IDirect3DRMDevice2::GetRenderMode

Retrieves the current transparency flags.

DWORD GetRenderMode( );

Return Values Returns the value of the current transparency flags.

Remarks Transparency flags have the following values:

Flag	Value
No flag set (default)	0
D3DRMRENDERMODE_BLENDEDTRANSPARENCY	1
D3DRMRENDERMODE_SORTEDTRANSPARENCY	2

See Also **IDirect3DRMDevice2::SetRenderMode**

IDirect3DRMDevice2::GetShades

Retrieves the number of shades in a ramp of colors used for shading.

DWORD GetShades( );

Return Values Returns the number of shades.

See Also **IDirect3DRMDevice2::SetShades**

IDirect3DRMDevice2::GetTextureQuality

Retrieves the current texture quality parameter for the device. Texture quality is relevant only for an RGB device.

D3DRMTEXTUREQUALITY GetTextureQuality( );

Return Values Returns one of the members of the **D3DRMTEXTUREQUALITY** enumerated type.

See Also **IDirect3DRMDevice2::SetTextureQuality**

IDirect3DRMDevice2::GetTrianglesDrawn

Retrieves the number of triangles drawn to a device since its creation. This method is a convenience function.

DWORD GetTrianglesDrawn( );

Return Values Returns the number of triangles.

Remarks The number of triangles includes those that were passed to the renderer but were not drawn because they were backfacing. The number does not include triangles that were rejected for lying outside of the viewing frustum.

IDirect3DRMDevice2::GetViewports

Constructs a Direct3DRMViewportArray object that represents the viewports currently constructed from the device.

HRESULT GetViewports(
LPDIRECT3DRMVIEWPORTARRAY* *lplpViewports*
);

Parameters *lplpViewports*
Address of a pointer that is initialized with a valid Direct3DRMViewportArray object if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice2::GetWidth

Retrieves the width, in pixels, of a device. This method is a convenience function.

DWORD GetWidth( );

Return Values Returns the width.

IDirect3DRMDevice2::GetWireframeOptions

Retrieves the wireframe options of a given device.

DWORD GetWireframeOptions( );

Return Values

Returns a bitwise **OR** of the following values:

D3DRMWIREFRAME_CULL

The backfacing faces are not drawn.

D3DRMWIREFRAME_HIDDENLINE

Wireframe-rendered lines are obscured by nearer objects.

IDirect3DRMDevice2::Init

Not implemented on the Windows platform.

HRESULT Init(
ULONG *width*,
ULONG *height*
);

IDirect3DRMDevice2::InitFromClipper

Initializes an **IDirect3DDevice2** device from a specified DirectDrawClipper object using **IDirect3DRM2::CreateDevice**.

HRESULT InitFromClipper(
LPDIRECTDRAWCLIPPER *lpDDClipper*,
LPGUID *lpGUID*,
int *width*,
int *height*
);

Parameters

lpDDClipper
Address of the DirectDrawClipper object to use as an initializer.

lpGUID the Direct3D device driver to use.
width and *height*
Width and height of the device.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMDevice2::InitFromD3D2

Initializes an **IDirect3DRMDevice2** Retained-Mode device from an **IDirect3D2** Immediate-Mode object and an **IDirect3DDevice2** Immediate-Mode device.

HRESULT InitFromD3D2(
LPDIRECT3D2 *lpD3D*,
LPDIRECT3DDEVICE2 *lpD3DIMDev*
);

Parameters

lpD3D
Address of the **IDirect3D2** Immediate-Mode object to use to initialize the Retained-Mode device.

lpD3DIMDev
Address of the **IDirect3DDevice2** Immediate-Mode device to use to initialize the Retained-Mode device.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The **IDirect3DRMDevice2** device initialized from **IDirect3DDevice2** will support the DrawPrimitive interface and execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRMDevice2::InitFromSurface

Initializes a **IDirect3DDevice2** device from a specified DirectDraw surface, using the **IDirect3DRM2::CreateDevice** method.

HRESULT InitFromSurface(
LPGUID *lpGUID*,
LPDIRECTDRAW *lpDD*,
LPDIRECTDRAWSURFACE *lpDDSBack*

);

Parameters

lpGUID
Address of the globally unique identifier (GUID) that identifies the Direct3D device driver to use.

lpDD
Address of the interface of a DirectDraw object that created the DirectDrawSurface.

lpDDSBack
Address of the interface of a DirectDrawSurface back buffer onto which the device will render.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The **IDirect3DRMDevice2** device initialized will support the DrawPrimitive interface and execute buffers, and is required for progressive meshes and for alpha blending and sorting of transparent objects.

IDirect3DRMDevice2::SetBufferCount

Sets the number of buffers currently being used by the application.

HRESULT SetBufferCount(
DWORD *dwCount*
);

Parameters

dwCount
Specifies the number of buffers—one for single-buffering, two for double-buffering, and so on. The default value is 1, which is correct only for single-buffered window operation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An application that employs double-buffering or triple-buffering must use this method to inform the system of how many buffers it is using so that the system can calculate how much of the window to clear and update on each frame.

See Also

IDirect3DRMDevice2::GetBufferCount

IDirect3DRMDevice2::SetDither

Sets the dither flag for the device.

HRESULT SetDither(
BOOL *bDither*
);

Parameters

bDither
New dithering mode for the device. The default is TRUE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMDevice2::GetDither

IDirect3DRMDevice2::SetQuality

Sets the rendering quality of a device

HRESULT SetQuality (
D3DRMRENDERQUALITY *rqQuality*
);

Parameters

rqQuality
One or more of the members of the enumerated types represented by the **D3DRMRENDERQUALITY** type. The default setting is D3DRMRENDER_FLAT.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The rendering quality is the maximum quality at which rendering can take place on the rendering surface of that device.

An object's quality has three components: shade mode (flat or gouraud, phong is not yet implemented and will default to gouraud shading), lighting type (on or off), and fill mode (point, wireframe or solid).

You set the quality of a device with **SetQuality**. By default it is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMProgressiveMesh, Direct3DRMMeshBuilder, or Direct3DRMMeshBuilder2 object with their respective **SetQuality** methods. By default, the quality of these objects is D3DRMRENDER_GOURAUD (gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

See Also **IDirect3DRMDevice2::GetQuality**

IDirect3DRMDevice2::SetRenderMode

Sets the transparency mode. The mode type determines how transparent objects will be rendered. The default mode renders transparent objects with stippled transparency.

HRESULT SetRenderMode(
DWORD *dwFlags*
);

Parameters

dwFlags

One or more of the transparent mode flags. The default (*dwFlags* = 0) sets the transparency mode to stippled transparency. In addition, flags can have one or more of the following values:

D3DRMRENDERMODE_BLENDEDTRANSPARENCY (*dwFlags* = 1) sets the transparency mode to alpha blending.

D3DRMRENDERMODE_SORTEDTRANSPARENCY (*dwFlags* = 2) sets the transparency mode so that transparent polygons in the scene are buffered, sorted,

and rendered in a second pass. This flag has no effect if the D3DRMRENDERMODE_BLENDEDTRANSPARENCY flag is not also set.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks If the D3DRMRENDERMODE_BLENDEDTRANSPARENCY and D3DRMRENDERMODE_SORTEDTRANSPARENCY flags are set together, it ensures that when two transparent objects are rendered one on top of the other, the image will blend in the correct order to ensure the right visual result.

See Also **IDirect3DRMDevice2::GetRenderMode**

IDirect3DRMDevice2::SetShades

Sets the number of shades in a ramp of colors used for shading.

HRESULT SetShades(
DWORD *ulShades*
);

Parameters *ulShades*
New number of shades. This parameter must be a power of 2. The default is 32.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice2::GetShades**

IDirect3DRMDevice2::SetTextureQuality

Sets the texture quality for the device.

HRESULT SetTextureQuality(
D3DRMTEXTUREQUALITY *tqTextureQuality*
);

Parameters *tqTextureQuality*
One of the members of the **D3DRMTEXTUREQUALITY** enumerated type. The default is **D3DRMTEXTURE_NEAREST**.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMDevice2::GetTextureQuality**

IDirect3DRMDevice2::Update

Copies the image that has been rendered to the display. It also provides a heartbeat function to the device driver.

HRESULT Update( );

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks Each call to this method causes the system to call an application-defined callback function, **D3DRMUPDATECALLBACK**. To add a callback function, use the **IDirect3DRMDevice2::AddUpdateCallback** method.

See Also **IDirect3DRMDevice2::AddUpdateCallback**, **D3DRMUPDATECALLBACK**

IDirect3DRMFace

Applications use the methods of the **IDirect3DRMFace** interface to interact with a single polygon in a mesh. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMFace and IDirect3DRMFaceArray Interfaces*.

The methods of the **IDirect3DRMFace** interface can be organized into the following groups:

Color	**GetColor**
	SetColor
	SetColorRGB
Materials	**GetMaterial**
	SetMaterial
Textures	**GetTexture**
	GetTextureCoordinateIndex
	GetTextureCoordinates

	GetTextureTopology
	SetTexture
	SetTextureCoordinates
	SetTextureTopology
Vertices and normals	**AddVertex**
	AddVertexAndNormalIndexed
	GetNormal
	GetVertex
	GetVertexCount
	GetVertexIndex
	GetVertices

The **IDirect3DRMFace** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef
QueryInterface
Release

In addition, the **IDirect3DRMFace** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback
Clone
DeleteDestroyCallback
GetAppData
GetClassName
GetName
SetAppData
SetName

The Direct3DRMFace object is obtained by calling the **IDirect3DRM::CreateFace** method.

IDirect3DRMFace::AddVertex

Adds a vertex to a Direct3DRMFace object.

HRESULT AddVertex(
D3DVALUE *x*,
D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

x, y, and *z*
The x, y, and z components of the position of the new vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::AddVertexAndNormalIndexed

Adds a vertex and a normal to a Direct3DRMFace object, using an index for the vertex and an index for the normal in the containing mesh builder. The face, vertex, and normal must already be part of a Direct3DRMMeshBuilder object.

HRESULT AddVertexAndNormalIndexed(
DWORD *vertex*,
DWORD *normal*
);

Parameters

vertex and *normal*
Indexes of the vertex and normal to add.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::GetColor

Retrieves the color of a Direct3DRMFace object.

D3DCOLOR GetColor( );

Return Values

Returns the color.

See Also

IDirect3DRMFace::SetColor

IDirect3DRMFace::GetMaterial

Retrieves the material of a Direct3DRMFace object.

HRESULT GetMaterial(
LPDIRECT3DRMMATERIAL* *lplpMaterial*
);

Parameters

lplpMaterial
Address of a variable that will be filled with a pointer to the Direct3DRMMaterial object applied to the face.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFace::SetMaterial

IDirect3DRMFace::GetNormal

Retrieves the normal vector of a Direct3DRMFace object.

HRESULT GetNormal(
D3DVECTOR **lpNormal*
);

Parameters

lpNormal
Address of a **D3DVECTOR** structure that will be filled with the normal vector of the face.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::GetTexture

Retrieves the Direct3DRMTexture object applied to a Direct3DRMFace object.

HRESULT GetTexture(
LPDIRECT3DRMTEXTURE* *lplpTexture*

```
);
```

Parameters

lplpTexture
Address of a variable that will be filled with a pointer to the texture applied to the face.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFace::SetTexture

IDirect3DRMFace::GetTextureCoordinateIndex

Retrieves the index of the vertex for texture coordinates in the face's mesh. This index corresponds to the index specified in the *dwIndex* parameter.

```
int GetTextureCoordinateIndex(
    DWORD dwIndex
    );
```

Parameters

dwIndex
Index within the face of the vertex.

Return Values

Returns the index.

IDirect3DRMFace::GetTextureCoordinates

Retrieves the texture coordinates of a vertex in a Direct3DRMFace object.

```
HRESULT GetTextureCoordinates(
    DWORD index,
    D3DVALUE *lpU,
    D3DVALUE *lpV
    );
```

Parameters

index
Index of the vertex.

lpU and *lpV*
Addresses of variables that are filled with the texture coordinates of the vertex.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::GetTextureTopology

Retrieves the texture topology of a Direct3DRMFace object.

```
HRESULT GetTextureTopology(
    BOOL *lpU,
    BOOL *lpV
    );
```

Parameters *lpU* and *lpV*
Addresses of variables that are set or cleared depending on how the cylindrical wrapping flags are set for the face.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFace::SetTextureTopology**

IDirect3DRMFace::GetVertex

Retrieves the position and normal of a vertex in a Direct3DRMFace object.

```
HRESULT GetVertex(
    DWORD index,
    D3DVECTOR *lpPosition,
    D3DVECTOR *lpNormal
    );
```

Parameters *index*
Index of the vertex.

lpPosition and *lpNormal*
Addresses of **D3DVECTOR** structures that will be filled with the position and normal of the vertex, respectively.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::GetVertexCount

Retrieves the number of vertices in a Direct3DRMFace object.

int GetVertexCount( );

Return Values Returns the number of vertices.

IDirect3DRMFace::GetVertexIndex

Retrieves the index of the vertex in the face's mesh. This index corresponds to the index specified in the *dwIndex* parameter.

int GetVertexIndex (
DWORD *dwIndex*
);

Parameters *dwIndex*
Index within the face of the vertex.

Return Values Returns the index.

IDirect3DRMFace::GetVertices

Retrieves the position and normal vector of each vertex in a Direct3DRMFace object.

HRESULT GetVertices(
DWORD **lpdwVertexCount*,
D3DVECTOR **lpPosition*,
D3DVECTOR **lpNormal*
);

Parameters

lpdwVertexCount
Address of a variable that is filled with the number of vertices. This parameter cannot be NULL.

lpPosition and *lpNormal*
Arrays of **D3DVECTOR** structures that will be filled with the positions and normal vectors of the vertices, respectively. If both of these parameters are NULL, the method will fill the *lpdwVertexCount* parameter with the number of vertices that will be retrieved.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::SetColor

Sets a Direct3DRMFace object to a given color.

HRESULT SetColor(
D3DCOLOR *color*
);

Parameters

color
Color to set.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFace::GetColor

IDirect3DRMFace::SetColorRGB

Sets a Direct3DRMFace object to a given color.

HRESULT SetColorRGB(
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*
);

Parameters

red, green, and *blue*
The red, green, and blue components of the color.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::SetMaterial

Sets the material of a Direct3DRMFace object.

HRESULT SetMaterial(
LPDIRECT3DRMMATERIAL *lpD3DRMMaterial*
);

Parameters *lpD3DRMMaterial*
Address of the material.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFace::GetMaterial**

IDirect3DRMFace::SetTexture

Sets the texture of a Direct3DRMFace object.

HRESULT SetTexture(
LPDIRECT3DRMTEXTURE *lpD3DRMTexture*
);

Parameters *lpD3DRMTexture*
Address of the texture.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFace::GetTexture**

IDirect3DRMFace::SetTextureCoordinates

Sets the texture coordinates of a specified vertex in a Direct3DRMFace object.

HRESULT SetTextureCoordinates(
DWORD *vertex*,
D3DVALUE *u*,
D3DVALUE *v*
);

Parameters

vertex
Index of the vertex to be set. For example, if the face were a triangle, the possible vertex indices would be 0, 1, and 2.

u and *v*
Texture coordinates to assign to the specified vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFace::SetTextureTopology

Sets the texture topology of a Direct3DRMFace object.

HRESULT SetTextureTopology(
BOOL *cylU*,
BOOL *cylV*
);

Parameters

cylU and *cylV*
Specify whether the texture has a cylindrical topology in the u and v dimensions.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFace::GetTextureTopology

IDirect3DRMFrame

Applications use the methods of the **IDirect3DRMFrame** interface to interact with frames—an object's frame of reference. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMFrame, IDirect3DRMFrame2, and IDirect3DRMFrameArray Interfaces*.

The methods of the **IDirect3DRMFrame** interface can be organized into the following groups:

Background	**GetSceneBackground**

	GetSceneBackgroundDepth
	SetSceneBackground
	SetSceneBackgroundDepth
	SetSceneBackgroundImage
	SetSceneBackgroundRGB
Color	**GetColor**
	SetColor
	SetColorRGB
Fog	**GetSceneFogColor**
	GetSceneFogEnable
	GetSceneFogMode
	GetSceneFogParams
	SetSceneFogColor
	SetSceneFogEnable
	SetSceneFogMode
	SetSceneFogParams
Hierarchies	**AddChild**
	DeleteChild
	GetChildren
	GetParent
	GetScene
Lighting	**AddLight**
	DeleteLight
	GetLights
Loading	**Load**
Material modes	**GetMaterialMode**
	SetMaterialMode
Positioning and movement	**AddMoveCallback**
	AddRotation

	AddScale
	AddTranslation
	DeleteMoveCallback
	GetOrientation
	GetPosition
	GetRotation
	GetVelocity
	LookAt
	Move
	SetOrientation
	SetPosition
	SetRotation
	SetVelocity
Sorting	**GetSortMode**
	GetZbufferMode
	SetSortMode
	SetZbufferMode
Textures	**GetTexture**
	GetTextureTopology
	SetTexture
	SetTextureTopology
Transformations	**AddTransform**
	GetTransform
	InverseTransform
	Transform
Visual objects	**AddVisual**
	DeleteVisual
	GetVisuals

The **IDirect3DRMFrame** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMFrame** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMFrame object is obtained by calling the **IDirect3DRM::CreateFrame** method.

IDirect3DRMFrame::AddChild

Adds a child frame to a frame hierarchy.

```
HRESULT AddChild(
    LPDIRECT3DRMFRAME lpD3DRMFrameChild
    );
```

Parameters

lpD3DRMFrameChild
Address of the Direct3DRMFrame object that will be added as a child.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the frame being added as a child already has a parent, this method removes it from its previous parent before adding it to the new parent.

To preserve an object's transformation, use the **IDirect3DRMFrame::GetTransform** method to retrieve the object's transformation before using the **AddChild** method. Then reapply the transformation after the frame is added.

IDirect3DRMFrame::AddLight

Adds a light to a frame.

HRESULT AddLight(
LPDIRECT3DRMLIGHT *lpD3DRMLight*
);

Parameters

lpD3DRMLight
Address of a variable that represents the Direct3DRMLight object to be added to the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::AddMoveCallback

Adds a callback function for special movement processing.

HRESULT AddMoveCallback(
D3DRMFRAMEMOVECALLBACK *d3drmFMC*,
VOID * *lpArg*
);

Parameters

d3drmFMC
Application-defined **D3DRMFRAMEMOVECALLBACK** callback function.

lpArg
Application-defined data to be passed to the callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::Move, **IDirect3DRMFrame::DeleteMoveCallback**

IDirect3DRMFrame::AddRotation

Adds a rotation about (*rvX*, *rvY*, *rvZ*) by the number of radians specified in *rvTheta*.

HRESULT AddRotation(
D3DRMCOMBINETYPE *rctCombine*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*,
D3DVALUE *rvTheta*
);

Parameters

rctCombine
A member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new rotation with any current frame transformation.

rvX, rvY, and *rvZ*
Axis about which to rotate.

rvTheta
Angle of rotation, in radians.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The specified rotation changes the matrix only for the frame identified by this *IDirect3DRMFrame* interface. This method changes the objects in the frame only once, unlike **IDirect3DRMFrame::SetRotation**, which changes the matrix with every render tick.

IDirect3DRMFrame::AddScale

Scales a frame's local transformation by (*rvX*, *rvY*, *rvZ*).

HRESULT AddScale(
D3DRMCOMBINETYPE *rctCombine*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new scale with any current frame transformation.

rvX, rvY, and *rvZ*
Define the scale factors in the x, y, and z directions.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The specified transformation changes the matrix only for the frame identified by this *IDirect3DRMFrame* interface.

IDirect3DRMFrame::AddTransform

Transforms the local coordinates of the frame by the given affine transformation according to the value of the *rctCombine* parameter.

HRESULT AddTransform(
D3DRMCOMBINETYPE *rctCombine*,
D3DRMMATRIX4D *rmMatrix*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new transformation with any current transformation.

rmMatrix
Member of the **D3DRMMATRIX4D** array that defines the transformation matrix to be combined.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks Although a 4×4 matrix is given, the last column must be the transpose of [0 0 0 1] for the transformation to be affine.

The specified transformation changes the matrix only for the frame identified by this *IDirect3DRMFrame* interface.

IDirect3DRMFrame::AddTranslation

Adds a translation by (*rvX*, *rvY*, *rvZ*) to a frame's local coordinate system.

HRESULT AddTranslation(
D3DRMCOMBINETYPE *rctCombine*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new translation with any current translation.

rvX, rvY, and *rvZ*
Define the position changes in the x, y, and z directions.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The specified translation changes the matrix only for the frame identified by this *IDirect3DRMFrame* interface.

IDirect3DRMFrame::AddVisual

Adds a visual object to a frame.

HRESULT AddVisual(
LPDIRECT3DRMVISUAL *lpD3DRMVisual*
);

Parameters

lpD3DRMVisual
Address of a variable that represents the Direct3DRMVisual object to be added to the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Visual objects include meshes and textures. When a visual object is added to a frame, it becomes visible if the frame is in view. The visual object is referenced by the frame.

IDirect3DRMFrame::DeleteChild

Removes a frame from the hierarchy. If the frame is not referenced, it is destroyed along with any child frames, lights, and meshes.

HRESULT DeleteChild(
LPDIRECT3DRMFRAME *lpChild*
);

Parameters *lpChild*
Address of a variable that represents the Direct3DRMFrame object to be used as the child.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::DeleteLight

Removes a light from a frame, destroying it if it is no longer referenced. When a light is removed from a frame, it no longer affects meshes in the scene its frame was in.

HRESULT DeleteLight(
LPDIRECT3DRMLIGHT *lpD3DRMLight*
);

Parameters *lpD3DRMLight*
Address of a variable that represents the Direct3DRMLight object to be removed.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::DeleteMoveCallback

Removes a callback function that performed special movement processing.

HRESULT DeleteMoveCallback(
D3DRMFRAMEMOVECALLBACK *d3drmFMC*,
VOID * *lpArg*
);

Parameters *d3drmFMC*
Application-defined **D3DRMFRAMEMOVECALLBACK** callback function.

lpArg
Application-defined data that was passed to the callback function.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame::AddMoveCallback**, **IDirect3DRMFrame::Move**

IDirect3DRMFrame::DeleteVisual

Removes a visual object from a frame, destroying it if it is no longer referenced.

HRESULT DeleteVisual(
LPDIRECT3DRMVISUAL *lpD3DRMVisual*
);

Parameters

lpD3DRMVisual
Address of a variable that represents the Direct3DRMVisual object to be removed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetChildren

Retrieves a list of child frames in the form of a Direct3DRMFrameArray object.

HRESULT GetChildren(
LPDIRECT3DRMFRAMEARRAY* *lplpChildren*
);

Parameters

lplpChildren
Address of a pointer to be initialized with a valid Direct3DRMFrameArray pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

Direct3DRMFrameArray, *Hierarchies*

IDirect3DRMFrame::GetColor

Retrieves the color of the frame.

D3DCOLOR GetColor( );

Return Values

Returns the color of the Direct3DRMFrame object.

See Also **IDirect3DRMFrame::SetColor**

IDirect3DRMFrame::GetLights

Retrieves a list of lights in the frame in the form of a Direct3DRMLightArray object.

HRESULT GetLights(
LPDIRECT3DRMLIGHTARRAY* *lplpLights*
);

Parameters

lplpLights
Address of a pointer to be initialized with a valid Direct3DRMLightArray pointer if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetMaterialMode

Retrieves the material mode of the frame.

D3DRMMATERIALMODE GetMaterialMode( );

Return Values Returns a member of the **D3DRMMATERIALMODE** enumerated type that specifies the current material mode.

See Also **IDirect3DRMFrame::SetMaterialMode**

IDirect3DRMFrame::GetOrientation

Retrieves the orientation of a frame relative to the given reference frame.

HRESULT GetOrientation(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvDir*,
LPD3DVECTOR *lprvUp*

);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvDir and *lprvUp*
Addresses of **D3DVECTOR** structures that will be filled with the directions of the frame's z-axis and y-axis, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::SetOrientation

IDirect3DRMFrame::GetParent

Retrieves the parent frame of the current frame.

HRESULT GetParent(
LPDIRECT3DRMFRAME* *lplpParent*
);

Parameters

lplpParent
Address of a pointer that will be filled with the pointer to the Direct3DRMFrame object representing the frame's parent. If the current frame is the root, this pointer will be NULL when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetPosition

Retrieves the position of a frame relative to the given reference frame (for example, this method retrieves the distance of the frame from the reference). The distance is stored in the *lprvPos* parameter as a vector rather than as a linear measure.

HRESULT GetPosition(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvPos*

);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvPos
Address of a **D3DVECTOR** structure that will be filled with the frame's position.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame::SetPosition**

IDirect3DRMFrame::GetRotation

Retrieves the rotation of the frame relative to the given reference frame.

HRESULT GetRotation(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvAxis*,
LPD3DVALUE *lprvTheta*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvAxis
Address of a **D3DVECTOR** structure that will be filled with the frame's axis of rotation.

lprvTheta
Address of a variable that will be the frame's rotation, in radians.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame::SetRotation**, *Transformations*

IDirect3DRMFrame::GetScene

Retrieves the root frame of the hierarchy containing the given frame.

HRESULT GetScene(
LPDIRECT3DRMFRAME* *lplpRoot*
);

Parameters

lplpRoot
Address of the pointer that will be filled with the pointer to the Direct3DRMFrame object representing the scene's root frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetSceneBackground

Retrieves the background color of a scene.

D3DCOLOR GetSceneBackground( );

Return Values

Returns the color.

IDirect3DRMFrame::GetSceneBackgroundDepth

Retrieves the current background-depth buffer for the scene.

HRESULT GetSceneBackgroundDepth(
LPDIRECTDRAWSURFACE * *lplpDDSurface*
);

Parameters

lplpDDSurface
Address of a pointer that will be initialized with the address of a DirectDraw surface representing the current background-depth buffer.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::SetSceneBackgroundDepth

IDirect3DRMFrame::GetSceneFogColor

Retrieves the fog color of a scene.

D3DCOLOR GetSceneFogColor( );

Return Values Returns the fog color.

IDirect3DRMFrame::GetSceneFogEnable

Returns whether fog is currently enabled for this scene.

BOOL GetSceneFogEnable( );

Return Values Returns TRUE if fog is enabled, and FALSE otherwise.

IDirect3DRMFrame::GetSceneFogMode

Returns the current fog mode for this scene.

D3DRMFOGMODE GetSceneFogMode( );

Return Values Returns a member of the **D3DRMFOGMODE** enumerated type that specifies the current fog mode.

IDirect3DRMFrame::GetSceneFogParams

Retrieves the current fog parameters for this scene.

HRESULT GetSceneFogParams(
D3DVALUE * *lprvStart*,
D3DVALUE * *lprvEnd*,
D3DVALUE * *lprvDensity*

);

Parameters *lprvStart, lprvEnd,* and *lprvDensity*
Addresses of variables that will be the fog start, end, and density values.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetSortMode

Retrieves the sorting mode used to process child frames.

D3DRMSORTMODE GetSortMode( );

Return Values Returns the member of the **D3DRMSORTMODE** enumerated type that specifies the sorting mode.

See Also **IDirect3DRMFrame::SetSortMode**

IDirect3DRMFrame::GetTexture

Retrieves the texture of the given frame.

HRESULT GetTexture(
LPDIRECT3DRMTEXTURE* *lplpTexture*
);

Parameters *lplpTexture*
Address of the pointer that will be filled with the address of the Direct3DRMTexture object representing the frame's texture.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame::SetTexture**

IDirect3DRMFrame::GetTextureTopology

Retrieves the topological properties of a texture when mapped onto objects in the given frame.

HRESULT GetTextureTopology(
BOOL * *lpbWrap_u*,
BOOL * *lpbWrap_v*
);

Parameters

lpbWrap_u and *lpbWrap_v*
Addresses of variables that are set to TRUE if the texture is mapped in the u and v directions, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::SetTextureTopology

IDirect3DRMFrame::GetTransform

Retrieves the local transformation of the frame as a 4×4 affine matrix.

HRESULT GetTransform(
D3DRMMATRIX4D *rmMatrix*
);

Parameters

rmMatrix
A **D3DRMMATRIX4D** array that will be filled with the frame's transformation. Because this is an array, this value is actually an address.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetVelocity

Retrieves the velocity of the frame relative to the given reference frame.

HRESULT GetVelocity(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvVel*,
BOOL *fRotVel*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvVel
Address of a **D3DVECTOR** structure that will be filled with the frame's velocity.

fRotVel
Flag specifying whether the rotational velocity of the object is taken into account when retrieving the linear velocity. If this parameter is TRUE, the object's rotational velocity is included in the calculation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::SetVelocity

IDirect3DRMFrame::GetVisuals

Retrieves a list of visuals in the frame.

HRESULT GetVisuals(
LPDIRECT3DRMVISUALARRAY* *lplpVisuals*
);

Parameters

lplpVisuals
Address of a pointer to be initialized with a valid Direct3DRMVisualArray pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::GetZbufferMode

Retrieves the z-buffer mode; that is, whether z-buffering is enabled or disabled.

D3DRMZBUFFERMODE GetZbufferMode( );

Return Values Returns one of the members of the **D3DRMZBUFFERMODE** enumerated type.

See Also **IDirect3DRMFrame::SetZbufferMode**

IDirect3DRMFrame::InverseTransform

Transforms the vector in the *lprvSrc* parameter in world coordinates to model coordinates, and returns the result in the *lprvDst* parameter.

HRESULT InverseTransform(
D3DVECTOR **lprvDst*,
D3DVECTOR **lprvSrc*
);

Parameters

lprvDst
Address of a **D3DVECTOR** structure that will be filled with the result of the transformation.

lprvSrc
Address of a **D3DVECTOR** structure that is the source of the transformation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame::Transform**, *3D Transformations*

IDirect3DRMFrame::Load

Loads a Direct3DRMFrame object.

HRESULT Load(
LPVOID *lpvObjSource*,
LPVOID *lpvObjID*,
D3DRMLOADOPTIONS *d3drmLOFlags*,
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc*,
LPVOID *lpArgLTP*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by the object that require special formatting. This parameter can be NULL.

lpArgLTP
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

By default, this method loads the first frame hierarchy in the file specified by the *lpvObjSource* parameter. The frame that calls this method is used as the parent of the new frame hierarchy.

IDirect3DRMFrame::LookAt

Faces the frame toward the target frame, relative to the given reference frame, locking the rotation by the given constraints.

HRESULT LookAt(
LPDIRECT3DRMFRAME *lpTarget***,**
LPDIRECT3DRMFRAME *lpRef***,**
D3DRMFRAMECONSTRAINT *rfcConstraint*
);

Parameters

lpTarget and *lpRef*
Addresses of variables that represent the Direct3DRMFrame objects to be used as the target and reference, respectively.

rfcConstraint
Member of the **D3DRMFRAMECONSTRAINT** enumerated type that specifies the axis of rotation to constrain.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::Move

Applies the rotations and velocities for all frames in the given hierarchy.

HRESULT Move(
D3DVALUE *delta*
);

Parameters *delta*
Amount to change the velocity and rotation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::SetColor

Sets the color of the frame. This color is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME.

HRESULT SetColor(
D3DCOLOR *rcColor*
);

Parameters *rcColor*
New color for the frame.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a color key to a Direct3DRMFrameInterpolator object.

See Also **IDirect3DRMFrame::GetColor**, **IDirect3DRMFrame::SetMaterialMode**

IDirect3DRMFrame::SetColorRGB

Sets the color of the frame. This color is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME.

HRESULT SetColorRGB(
D3DVALUE *rvRed*,
D3DVALUE *rvGreen*,
D3DVALUE *rvBlue*
);

Parameters

rvRed, rvGreen, and *rvBlue*
New color for the frame. Each component of the color should be in the range 0 to 1.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add an RGB color key to a Direct3DRMFrameInterpolator object.

See Also

IDirect3DRMFrame::SetMaterialMode

IDirect3DRMFrame::SetMaterialMode

Sets the material mode for a frame. The material mode determines the source of material information for visuals rendered with the frame.

HRESULT SetMaterialMode(
D3DRMMATERIALMODE *rmmMode*
);

Parameters

rmmMode
One of the members of the **D3DRMMATERIALMODE** enumerated type.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::GetMaterialMode

IDirect3DRMFrame::SetOrientation

Aligns a frame so that its z-direction points along the direction vector [*rvDx, rvDy, rvDz*] and its y-direction aligns with the vector [*rvUx, rvUy, rvUz*].

HRESULT SetOrientation(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvDx*,
D3DVALUE *rvDy*,
D3DVALUE *rvDz*,
D3DVALUE *rvUx*,
D3DVALUE *rvUy*,
D3DVALUE *rvUz*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvDx, rvDy, and *rvDz*
New z-axis for the frame.

rvUx, rvUy, and *rvUz*
New y-axis for the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The default orientation of a frame has a direction vector of [0, 0, 1] and an up vector of [0, 1, 0].

If [*rvUx, rvUy, rvUz*] is parallel to [*rvDx, rvDy, rvDz*], the D3DRMERR_BADVALUE error value is returned; otherwise, the [*rvUx, rvUy, rvUz*] vector passed is projected onto the plane that is perpendicular to [*rvDx, rvDy, rvDz*].

This method is also used to add an orientation key to a Direct3DRMRFrameInterpolator object.

See Also

IDirect3DRMFrame::GetOrientation

IDirect3DRMFrame::SetPosition

Sets the position of a frame relative to the frame of reference. It places the frame a distance of [*rvX, rvY, rvZ*] from the reference. When a child frame is created within a parent, it is placed at [0, 0, 0] in the parent frame.

HRESULT SetPosition(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
New position for the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a position key to a Direct3DRMFrameInterpolator object.

See Also

IDirect3DRMFrame::GetPosition

IDirect3DRMFrame::SetRotation

Sets a frame rotating by the given angle around the given vector at each call to the **IDirect3DRM::Tick** or **IDirect3DRMFrame::Move** method. The direction vector [*rvX, rvY, rvZ*] is defined in the reference frame.

HRESULT SetRotation(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*,
D3DVALUE *rvTheta*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
Vector about which rotation occurs.

rvTheta
Rotation angle, in radians.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The specified rotation changes the matrix with every render tick, unlike the **IDirect3DRMFrame::AddRotation** method, which changes the objects in the frame only once.

See Also

IDirect3DRMFrame::AddRotation, **IDirect3DRMFrame::GetRotation**

IDirect3DRMFrame::SetSceneBackground

Sets the background color of a scene.

HRESULT SetSceneBackground(
D3DCOLOR *rcColor*
);

Parameters

rcColor
New color for the background.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a background color key to a Direct3DRMFrameInterpolator object.

IDirect3DRMFrame::SetSceneBackgroundDepth

Specifies a background-depth buffer for a scene.

HRESULT SetSceneBackgroundDepth(
LPDIRECTDRAWSURFACE *lpImage*

);

Parameters

lpImage
Address of a DirectDraw surface that will store the new background depth for the scene.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The image must have a depth of 16. If the image and viewport sizes are different, the image is scaled first. For best performance when animating the background-depth buffer, the image should be the same size as the viewport. This enables the depth buffer to be updated directly from the image memory without incurring extra overhead.

See Also

IDirect3DRMFrame::GetSceneBackgroundDepth

IDirect3DRMFrame::SetSceneBackgroundImage

Specifies a background image for a scene.

HRESULT SetSceneBackgroundImage(
LPDIRECT3DRMTEXTURE *lpTexture*
);

Parameters

lpTexture
Address of a Direct3DRMTexture object that will contain the new background scene.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the image is a different size or color depth than the viewport, the image will first be scaled or converted to the correct depth. For best performance when animating the background, the image should be the same size and color depth. This enables the background to be rendered directly from the image memory without incurring any extra overhead.

IDirect3DRMFrame::SetSceneBackgroundRGB

Sets the background color of a scene.

HRESULT SetSceneBackgroundRGB(
D3DVALUE *rvRed*,
D3DVALUE *rvGreen*,
D3DVALUE *rvBlue*
);

Parameters

rvRed, rvGreen, and *rvBlue*
New color for the background.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a background RGB color key to a Direct3DRMFrameInterpolator object.

IDirect3DRMFrame::SetSceneFogColor

Sets the fog color of a scene.

HRESULT SetSceneFogColor(
D3DCOLOR *rcColor*
);

Parameters

rcColor
New color for the fog.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a fog color key to a Direct3DRMRFrameInterpolator object.

IDirect3DRMFrame::SetSceneFogEnable

Sets the fog enable state.

HRESULT SetSceneFogEnable(
BOOL *bEnable*
);

Parameters

bEnable
New fog enable state.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame::SetSceneFogMode

Sets the fog mode.

HRESULT SetSceneFogMode(
D3DRMFOGMODE *rfMode*
);

Parameters

rfMode
One of the members of the **D3DRMFOGMODE** enumerated type, specifying the new fog mode.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::SetSceneFogParams

IDirect3DRMFrame::SetSceneFogParams

Sets the current fog parameters for this scene.

HRESULT SetSceneFogParams(
D3DVALUE *rvStart*,
D3DVALUE *rvEnd*,
D3DVALUE *rvDensity*

);

Parameters

rvStart and *rvEnd*
Fog start and end points for linear fog mode. These settings determine the distance from the camera at which fog effects first become visible and the distance at which fog reaches its maximum density.

rvDensity
Fog density for the exponential fog modes. This value should be in the range 0 through 1.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a fog parameters key to a Direct3DRMRFrameInterpolator object.

See Also

D3DRMFOGMODE, **IDirect3DRMFrame::SetSceneFogMode**

IDirect3DRMFrame::SetSortMode

Sets the sorting mode used to process child frames. You can use this method to change the properties of hidden-surface-removal algorithms.

HRESULT SetSortMode(
D3DRMSORTMODE *d3drmSM*
);

Parameters

d3drmSM
One of the members of the **D3DRMSORTMODE** enumerated type, specifying the sorting mode. The default value is D3DRMSORT_FROMPARENT.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::GetSortMode

IDirect3DRMFrame::SetTexture

Sets the texture of the frame.

HRESULT SetTexture(
LPDIRECT3DRMTEXTURE *lpD3DRMTexture*
);

Parameters

lpD3DRMTexture
Address of a variable that represents the Direct3DRMTexture object to be used.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The texture is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME. To disable the frame's texture, use a NULL texture.

See Also

IDirect3DRMFrame::GetTexture, **IDirect3DRMFrame::SetMaterialMode**

IDirect3DRMFrame::SetTextureTopology

Defines the topological properties of the texture coordinates across objects in the frame.

HRESULT SetTextureTopology(
BOOL *bWrap_u*,
BOOL *bWrap_v*
);

Parameters

bWrap_u and *bWrap_v*
Variables that are set to TRUE to map the texture in the u- and v-directions, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::GetTextureTopology

IDirect3DRMFrame::SetVelocity

Sets the velocity of the given frame relative to the reference frame. The frame will be moved by the vector [*rvX, rvY, rvZ*] with respect to the reference frame at

each successive call to the **IDirect3DRM::Tick** or **IDirect3DRMFrame::Move** method.

HRESULT SetVelocity(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*,
BOOL *fRotVel*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
New velocity for the frame.

fRotVel
Flag specifying whether the rotational velocity of the object is taken into account when setting the linear velocity. If TRUE, the object's rotational velocity is included in the calculation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::GetVelocity

IDirect3DRMFrame::SetZbufferMode

Sets the z-buffer mode; that is, whether z-buffering is enabled or disabled.

HRESULT SetZbufferMode(
D3DRMZBUFFERMODE *d3drmZBM*
);

Parameters

d3drmZBM
One of the members of the **D3DRMZBUFFERMODE** enumerated type, specifying the z-buffer mode. The default value is D3DRMZBUFFER_FROMPARENT.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::GetZbufferMode

IDirect3DRMFrame::Transform

Transforms the vector in the *lpd3dVSrc* parameter in model coordinates to world coordinates, returning the result in the *lpd3dVDst* parameter.

HRESULT Transform(
D3DVECTOR **lpd3dVDst*,
D3DVECTOR **lpd3dVSrc*
);

Parameters

lpd3dVDst
Address of a **D3DVECTOR** structure that will be filled with the result of the transformation operation.

lpd3dVSrc
Address of a **D3DVECTOR** structure that is the source of the transformation operation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame::InverseTransform, *3D Transformations*

IDirect3DRMFrame2

The **IDirect3DRMFrame2** interface is an extension of the **IDirect3DRMFrame** interface. **IDirect3DRMFrame2** has new methods that enable using materials, bounding boxes, and axes with frames. **IDirect3DRMFrame2** also has a **IDirect3DRMFrame2::RayPick** method to calculate the intersections of visuals in the frame with a ray of specified position and direction. In addition, **IDirect3DRMFrame2** has one changed method **IDirect3DRMFrame2::AddMoveCallback2**.

For a conceptual overview, see *IDirect3DRMFrame, IDirect3DRMFrame2, and IDirect3DRMFrameArray Interfaces*.

The methods of the **IDirect3DRMFrame2** interface can be organized into the following groups:

Axes	**GetAxes**
	GetInheritAxes
	SetAxes
	SetInheritAxes

Background	**GetSceneBackground**
	GetSceneBackgroundDepth
	SetSceneBackground
	SetSceneBackgroundDepth
	SetSceneBackgroundImage
	SetSceneBackgroundRGB
Bounding Box	**GetBox**
	GetBoxEnable
	GetHierarchyBox
	SetBox
	SetBoxEnable
Color	**GetColor**
	SetColor
	SetColorRGB
Fog	**GetSceneFogColor**
	GetSceneFogEnable
	GetSceneFogMode
	GetSceneFogParams
	SetSceneFogColor
	SetSceneFogEnable
	SetSceneFogMode
	SetSceneFogParams
Hierarchies	**AddChild**
	DeleteChild
	GetChildren
	GetParent
	GetScene
Lighting	**AddLight**
	DeleteLight
	GetLights
Loading	**Load**

Material	**GetMaterial**
	SetMaterial
Material modes	**GetMaterialMode**
	SetMaterialMode
Positioning and movement	**AddMoveCallback2**
	AddRotation
	AddScale
	AddTranslation
	DeleteMoveCallback
	GetOrientation
	GetPosition
	GetRotation
	GetVelocity
	LookAt
	Move
	SetOrientation
	SetPosition
	SetQuaternion
	SetRotation
	SetVelocity
Ray Picking	**RayPick**
Sorting	**GetSortMode**
	GetZbufferMode
	SetSortMode
	SetZbufferMode
Textures	**GetTexture**
	GetTextureTopology
	SetTexture
	SetTextureTopology
Transformations	**AddTransform**

	GetTransform
	InverseTransform
	Transform
Visual objects	**AddVisual**
	DeleteVisual
	GetVisuals

The **IDirect3DRMFrame2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMFrame2** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMFrame2 object is obtained by calling the **IDirect3DRM2::CreateFrame** method.

IDirect3DRMFrame2::AddChild

Adds a child frame to a frame hierarchy.

HRESULT AddChild(
    LPDIRECT3DRMFRAME *lpD3DRMFrameChild*

```
);
```

Parameters

lpD3DRMFrameChild
Address of the Direct3DRMFrame object that will be added as a child.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the frame being added as a child already has a parent, this method removes it from its previous parent before adding it to the new parent.

To preserve an object's transformation, use the **IDirect3DRMFrame2::GetTransform** method to retrieve the object's transformation before using the **AddChild** method. Then reapply the transformation after the frame is added.

IDirect3DRMFrame2::AddLight

Adds a light to a frame.

```
HRESULT AddLight(
    LPDIRECT3DRMLIGHT lpD3DRMLight
    );
```

Parameters

lpD3DRMLight
Address of a variable that represents the Direct3DRMLight object to be added to the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::AddMoveCallback2

Adds a callback function for special movement processing.

```
HRESULT AddMoveCallback2(
    D3DRMFRAME2MOVECALLBACK d3drmFMC,
    VOID * lpArg,
    DWORD dwFlags
    );
```

Parameters

d3drmFMC
Application-defined **D3DRMFRAME2MOVECALLBACK** callback function.

lpArg
Application-defined data to be passed to the callback function.

dwFlags
One of the following values:

D3DRMCALLBACK_PREORDER - the default. When **IDirect3DRMFrame2::Move** traverses the hierarchy, callbacks for a frame are called before any child frames are traversed.

D3DRMCALLBACK_POSTORDER - When **IDirect3DRMFrame2::Move** traverses the hierarchy, callbacks for a frame are called after the child frames are traversed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Multiple callbacks on an individual frame are called in the order that the callbacks were created.

See Also

IDirect3DRMFrame2::Move, **IDirect3DRMFrame2::DeleteMoveCallback**

IDirect3DRMFrame2::AddRotation

Adds a rotation about (*rvX*, *rvY*, *rvZ*) by the number of radians specified in *rvTheta*.

HRESULT AddRotation(
D3DRMCOMBINETYPE *rctCombine***,**
D3DVALUE *rvX***,**
D3DVALUE *rvY***,**
D3DVALUE *rvZ***,**
D3DVALUE *rvTheta*
);

Parameters

rctCombine
A member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new rotation with any current frame transformation.

rvX, rvY, and *rvZ*
Axis about which to rotate.

rvTheta
Angle of rotation, in radians.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The specified rotation changes the matrix only for the frame identified by this *IDirect3DRMFrame2* interface. This method changes the objects in the frame only once, unlike **IDirect3DRMFrame2::SetRotation**, which changes the matrix with every render tick.

IDirect3DRMFrame2::AddScale

Scales a frame's local transformation by (*rvX*, *rvY*, *rvZ*).

HRESULT AddScale(
D3DRMCOMBINETYPE *rctCombine***,**
D3DVALUE *rvX***,**
D3DVALUE *rvY***,**
D3DVALUE *rvZ*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new scale with any current frame transformation.

rvX, rvY, and *rvZ*
Define the scale factors in the x, y, and z directions.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The specified transformation changes the matrix only for the frame identified by this *IDirect3DRMFrame2* interface.

IDirect3DRMFrame2::AddTransform

Transforms the local coordinates of the frame by the given affine transformation according to the value of the *rctCombine* parameter.

HRESULT AddTransform(
D3DRMCOMBINETYPE *rctCombine***,**
D3DRMMATRIX4D *rmMatrix*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new transformation with any current transformation.

rmMatrix
Member of the **D3DRMMATRIX4D** array that defines the transformation matrix to be combined.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Although a 4×4 matrix is given, the last column must be the transpose of [0 0 0 1] for the transformation to be affine.

The specified transformation changes the matrix only for the frame identified by this *IDirect3DRMFrame2* interface.

IDirect3DRMFrame2::AddTranslation

Adds a translation by (*rvX*, *rvY*, *rvZ*) to a frame's local coordinate system.

HRESULT AddTranslation(
D3DRMCOMBINETYPE *rctCombine***,**
D3DVALUE *rvX***,**
D3DVALUE *rvY***,**
D3DVALUE *rvZ*
);

Parameters

rctCombine
Member of the **D3DRMCOMBINETYPE** enumerated type that specifies how to combine the new translation with any current translation.

rvX, rvY, and *rvZ*
Define the position changes in the x, y, and z directions.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The specified translation changes the matrix only for the frame identified by this *IDirect3DRMFrame2* interface.

IDirect3DRMFrame2::AddVisual

Adds a visual object to a frame.

HRESULT AddVisual(
LPDIRECT3DRMVISUAL *lpD3DRMVisual*
);

Parameters

lpD3DRMVisual
Address of a variable that represents the Direct3DRMVisual object to be added to the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Visual objects include meshes and textures. When a visual object is added to a frame, it becomes visible if the frame is in view. The visual object is referenced by the frame.

IDirect3DRMFrame2::DeleteChild

Removes a frame from the hierarchy. If the frame is not referenced, it is destroyed along with any child frames, lights, and meshes.

HRESULT DeleteChild(
LPDIRECT3DRMFRAME *lpChild*
);

Parameters

lpChild
Address of a variable that represents the Direct3DRMFrame object to be used as the child.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::DeleteLight

Removes a light from a frame, destroying it if it is no longer referenced. When a light is removed from a frame, it no longer affects meshes in the scene its frame was in.

HRESULT DeleteLight(
LPDIRECT3DRMLIGHT *lpD3DRMLight*
);

Parameters

lpD3DRMLight
Address of a variable that represents the Direct3DRMLight object to be removed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::DeleteMoveCallback

Removes a callback function that performed special movement processing.

HRESULT DeleteMoveCallback(
D3DRMFRAMEMOVECALLBACK *d3drmFMC*,
VOID * *lpArg*
);

Parameters

d3drmFMC
Application-defined **D3DRMFRAMEMOVECALLBACK** callback function.

lpArg
Application-defined data that was passed to the callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::AddMoveCallback2, **IDirect3DRMFrame2::Move**

IDirect3DRMFrame2::DeleteVisual

Removes a visual object from a frame, destroying it if it is no longer referenced.

HRESULT DeleteVisual(
LPDIRECT3DRMVISUAL *lpD3DRMVisual*
);

Parameters

lpD3DRMVisual
Address of a variable that represents the Direct3DRMVisual object to be removed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetAxes

Retrieves the vectors that are aligned with the direction (*rvDx*, *rvDy*, *rvDz*) and up (*rvUx*, *rvUy*, *rvUz*) vectors supplied to the **IDirect3DRMFrame2::SetOrientation** method.

HRESULT GetAxes(
LPD3DVECTOR *dir***,**
LPD3DVECTOR *up*
);

Parameters

dir
The z-axis for the frame. Default is (0,0,1).

up
The y-axis for the frame. Default is (0,1,0).

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method along with **IDirect3DRMFrame2::SetAxes** helps support both left-handed and right-handed coordinate systems. **SetAxes** allows you to specify that the negative z-axis represents the front of the object.

See Also

IDirect3DRMFrame2::SetAxes, **IDirect3DRMFrame2::GetInheritAxes**, **IDirect3DRMFrame2::SetInheritAxes**

IDirect3DRMFrame2::GetBox

Retrieves the bounding box containing a DIRECT3DRMFRAME2 object. The bounding box gives the minimum and maximum model coordinates in each dimension.

HRESULT GetBox(
D3DRMBOX * *lpD3DRMBox*
);

Parameters

lpD3DRMBox
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. Returns D3DRMERR_BOXNOTSET unless a valid bounding box has already been set on the frame. For a list of other possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method supports a bounding box on a frame for hierarchical culling. A valid bounding box must be set on the frame with **IDirect3DRMFrame2::SetBox**. For a bounding box to be enabled, the **IDirect3DRMFrame2::SetBoxEnable** method must be called to set the enable flag to TRUE. By default, the box enable flag is FALSE. There is no default bounding box.

See Also

IDirect3DRMFrame2::SetBox, **IDirect3DRMFrame2::SetBoxEnable**, **IDirect3DRMFrame2::GetBoxEnable**

IDirect3DRMFrame2::GetBoxEnable

Retrieves the flag that determines whether a bounding box is enabled for this Direct3DRMFrame2 object.

BOOL GetBoxEnable( );

Return Values

Returns TRUE if a bounding box is enabled, or FALSE if it is not enabled.

Remarks

For a bounding box to be enabled, the **IDirect3DRMFrame2::SetBoxEnable** flag must be called to set the enable flag to TRUE. By default, the box enable flag is FALSE.

See Also **IDirect3DRMFrame2::SetBoxEnable IDirect3DRMFrame2::GetBox**, **IDirect3DRMFrame2::SetBox**

IDirect3DRMFrame2::GetChildren

Retrieves a list of child frames in the form of a Direct3DRMFrameArray object.

HRESULT GetChildren(
LPDIRECT3DRMFRAMEARRAY* *lplpChildren*
);

Parameters

lplpChildren
Address of a pointer to be initialized with a valid Direct3DRMFrameArray pointer if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also *DIRECT3DRMFRAMEARRAY*, *Hierarchies*

IDirect3DRMFrame2::GetColor

Retrieves the color of the frame.

D3DCOLOR GetColor( );

Return Values Returns the color of the Direct3DRMFrame2 object.

See Also **IDirect3DRMFrame2::SetColor**

IDirect3DRMFrame2::GetHierarchyBox

Calculates a bounding box to contain all the geometry in the hierarchy rooted in this Direct3DRMFrame2 object.

HRESULT GetHierarchyBox(
D3DRMBOX * *lpD3DRMBox*

);

Parameters

lpD3DRMBox
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::GetBox, **IDirect3DRMFrame2::SetBox**, **IDirect3DRMFrame2::SetBoxEnable**, **IDirect3DRMFrame2::GetBoxEnable**

IDirect3DRMFrame2::GetInheritAxes

Retrieves the flag that indicates whether the axes for the frame are inherited from the parent frame.

BOOL GetInheritAxes( );

Return Values

Returns TRUE if the frame inherits axes (the default) and FALSE if the frame does not inherit axes.

Remarks

By default, the axes are inherited from the parent. If a frame is set to inherit from the parent and there is no parent, the frame acts as if it inherits from a parent with the default axes (direction=(0,0,1) and up=(0,1,0)).

This method and **IDirect3DRMFrame2::SetInheritAxes** method allow a single policy for axes to be set at the root of the hierarchy.

See Also

IDirect3DRMFrame2::SetInheritAxes, **IDirect3DRMFrame2::GetAxes**, **IDirect3DRMFrame2::SetAxes**

IDirect3DRMFrame2::GetLights

Retrieves a list of lights in the frame in the form of a Direct3DRMLightArray object.

HRESULT GetLights(
LPDIRECT3DRMLIGHTARRAY* *lplpLights*
);

Parameters *lplpLights*
Address of a pointer to be initialized with a valid Direct3DRMLightArray pointer if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetMaterial

Retrieves the material of the Direct3DRMFrame2 object.

```
HRESULT GetMaterial(
    LPDIRECT3DRMMATERIAL *lplpMaterial
    );
```

Parameters *lplpMaterial*
Address of a variable that will be filled with a pointer to the Direct3DRMMaterial object applied to the frame.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::SetMaterial**

IDirect3DRMFrame2::GetMaterialMode

Retrieves the material mode of the frame.

```
D3DRMMATERIALMODE GetMaterialMode( );
```

Return Values Returns a member of the **D3DRMMATERIALMODE** enumerated type that specifies the current material mode.

See Also **IDirect3DRMFrame2::SetMaterialMode**

IDirect3DRMFrame2::GetOrientation

Retrieves the orientation of a frame relative to the given reference frame.

HRESULT GetOrientation(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvDir*,
LPD3DVECTOR *lprvUp*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvDir and *lprvUp*
Addresses of **D3DVECTOR** structures that will be filled with the normalized directions of the frame's z-axis and y-axis, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetOrientation

IDirect3DRMFrame2::GetParent

Retrieves the parent frame of the current frame.

HRESULT GetParent(
LPDIRECT3DRMRAME * *lplpParent*
);

Parameters

lplpParent
Address of a pointer that will be filled with the pointer to the Direct3DRMFrame object representing the frame's parent. If the current frame is the root, this pointer will be NULL when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetPosition

Retrieves the position of a frame relative to the given reference frame (for example, this method retrieves the distance of the frame from the reference). The distance is stored in the *lprvPos* parameter as a vector rather than as a linear measure.

HRESULT GetPosition(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvPos*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvPos
Address of a **D3DVECTOR** structure that will be filled with the frame's position.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetPosition

IDirect3DRMFrame2::GetRotation

Retrieves the rotation of the frame relative to the given reference frame.

HRESULT GetRotation(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvAxis*,
LPD3DVALUE *lprvTheta*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvAxis
Address of a **D3DVECTOR** structure that will be filled with the frame's axis of rotation.

lprvTheta
Address of a variable that will be the frame's rotation, in radians.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetRotation, *Transformations*

IDirect3DRMFrame2::GetScene

Retrieves the root frame of the hierarchy containing the given frame.

HRESULT GetScene(
LPDIRECT3DRMFRAME *lplpRoot*
);

Parameters

lplpRoot
Address of the pointer that will be filled with the pointer to the Direct3DRMFrame object representing the scene's root frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetSceneBackground

Retrieves the background color of a scene.

D3DCOLOR GetSceneBackground( );

Return Values

Returns the color.

IDirect3DRMFrame2::GetSceneBackgroundDepth

Retrieves the current background-depth buffer for the scene.

HRESULT GetSceneBackgroundDepth(
LPDIRECTDRAWSURFACE * *lplpDDSurface*
);

Parameters

lplpDDSurface
Address of a pointer that will be initialized with the address of a DirectDraw surface representing the current background-depth buffer.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::SetSceneBackgroundDepth**

IDirect3DRMFrame2::GetSceneFogColor

Retrieves the fog color of a scene.

D3DCOLOR GetSceneFogColor( );

Return Values Returns the fog color.

IDirect3DRMFrame2::GetSceneFogEnable

Returns whether fog is currently enabled for this scene.

BOOL GetSceneFogEnable( );

Return Values Returns TRUE if fog is enabled, and FALSE otherwise.

IDirect3DRMFrame2::GetSceneFogMode

Returns the current fog mode for this scene.

D3DRMFOGMODE GetSceneFogMode( );

Return Values Returns a member of the **D3DRMFOGMODE** enumerated type that specifies the current fog mode.

IDirect3DRMFrame2::GetSceneFogParams

Retrieves the current fog parameters for this scene.

HRESULT GetSceneFogParams(
D3DVALUE * *lprvStart*,
D3DVALUE * *lprvEnd*,
D3DVALUE * *lprvDensity*
);

Parameters

lprvStart, lprvEnd, and *lprvDensity*
Addresses of variables that will be the fog start, end, and density values.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetSortMode

Retrieves the sorting mode used to process child frames.

D3DRMSORTMODE GetSortMode( );

Return Values

Returns the member of the **D3DRMSORTMODE** enumerated type that specifies the sorting mode.

See Also

IDirect3DRMFrame2::SetSortMode

IDirect3DRMFrame2::GetTexture

Retrieves the texture of the given frame.

HRESULT GetTexture(
LPDIRECT3DRMTEXTURE* *lplpTexture*
);

Parameters

lplpTexture
Address of the pointer that will be filled with the address of the Direct3DRMTexture object representing the frame's texture.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetTexture

IDirect3DRMFrame2::GetTextureTopology

Retrieves the topological properties of a texture when mapped onto objects in the given frame.

HRESULT GetTextureTopology(
BOOL * *lpbWrap_u*,
BOOL * *lpbWrap_v*
);

Parameters

lpbWrap_u and *lpbWrap_v*
Addresses of variables that are set to TRUE if the texture is mapped in the u and v directions, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetTextureTopology

IDirect3DRMFrame2::GetTransform

Retrieves the local transformation of the frame as a 4×4 affine matrix.

HRESULT GetTransform(
D3DRMMATRIX4D *rmMatrix*
);

Parameters

rmMatrix
A **D3DRMMATRIX4D** array that will be filled with the frame's transformation. Because this is an array, this value is actually an address.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetVelocity

Retrieves the velocity of the frame relative to the given reference frame.

HRESULT GetVelocity(
LPDIRECT3DRMFRAME *lpRef*,
LPD3DVECTOR *lprvVel*,
BOOL *fRotVel*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

lprvVel
Address of a **D3DVECTOR** structure that will be filled with the frame's velocity.

fRotVel
Flag specifying whether the rotational velocity of the object is taken into account when retrieving the linear velocity. If this parameter is TRUE, the object's rotational velocity is included in the calculation.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetVelocity

IDirect3DRMFrame2::GetVisuals

Retrieves a list of visuals in the frame.

HRESULT GetVisuals(
LPDIRECT3DRMVISUALARRAY* *lplpVisuals*
);

Parameters

lplpVisuals
Address of a pointer to be initialized with a valid Direct3DRMVisualArray pointer if the call succeeds.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::GetZbufferMode

Retrieves the z-buffer mode; that is, whether z-buffering is enabled or disabled.

D3DRMZBUFFERMODE GetZbufferMode( );

Return Values Returns one of the members of the **D3DRMZBUFFERMODE** enumerated type.

See Also **IDirect3DRMFrame2::SetZbufferMode**

IDirect3DRMFrame2::InverseTransform

Transforms the vector in the *lprvSrc* parameter in world coordinates to model coordinates, and returns the result in the *lprvDst* parameter.

HRESULT InverseTransform(
D3DVECTOR **lprvDst*,
D3DVECTOR **lprvSrc*
);

Parameters

lprvDst
Address of a **D3DVECTOR** structure that will be filled with the result of the transformation.

lprvSrc
Address of a **D3DVECTOR** structure that is the source of the transformation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::Transform**, *3D Transformations*

IDirect3DRMFrame2::Load

Loads a Direct3DRMFrame2 object.

HRESULT Load(
LPVOID *lpvObjSource*,
LPVOID *lpvObjID*,
D3DRMLOADOPTIONS *d3drmLOFlags*,
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc*,
LPVOID *lpArgLTP*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by the object that require special formatting. This parameter can be NULL.

lpArgLTP
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

By default, this method loads the first frame hierarchy in the file specified by the *lpvObjSource* parameter. The frame that calls this method is used as the parent of the new frame hierarchy.

IDirect3DRMFrame2::LookAt

Faces the frame toward the target frame, relative to the given reference frame, locking the rotation by the given constraints.

HRESULT LookAt(
LPDIRECT3DRMFRAME *lpTarget***,**
LPDIRECT3DRMFRAME *lpRef***,**
D3DRMFRAMECONSTRAINT *rfcConstraint*
);

Parameters

lpTarget and *lpRef*
Addresses of variables that represent the Direct3DRMFrame objects to be used as the target and reference, respectively.

rfcConstraint
Member of the **D3DRMFRAMECONSTRAINT** enumerated type that specifies the axis of rotation to constrain.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::Move

Applies the rotations and velocities for all frames in the given hierarchy.

HRESULT Move(
D3DVALUE *delta*
);

Parameters *delta*
Amount to change the velocity and rotation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::RayPick

Searches the hierarchy starting at this Direct3DRMFrame2 object and calculates the intersections between any visuals and the ray specified by the *dvPosition* and *dvDirection* parameters in the coordinate space specified by the *lpRefFrame* parameter.

HRESULT RayPick(
LPDIRECT3DRMFRAME *lpRefFrame*,
LPD3DRMRAY *ray*,
DWORD *dwFlags*,
LPD3DRMPICKED2ARRAY* *lplpPicked2Array*
);

Parameters *lpRefFrame*
Address of the Direct3DRMFrame object that contains the ray.

ray
A pointer to a **D3DRMRAY** structure that contains two D3DVECTOR structures. The first D3DVECTOR structure is the vector direction of the ray. The second D3DVECTOR structure is the position of the origin of the ray.

dwFlags
Can be one of the following values:

D3DRMRAYPICK_ONLYBOUNDINGBOXES – Only intersections with bounding boxes of the visuals in the hierarchy are returned. Does not check for exact face intersections.

D3DRMRAYPICK_IGNOREFURTHERPRIMITIVES – Only the closest visual that intersects the ray is returned. Ignores visuals further away than the closest one found so far in a search.

D3DRMRAYPICK_INTERPOLATEUV – Interpolate texture coordinates.

D3DRMRAYPICK_INTERPOLATECOLOR – Interpolate color.

D3DRMRAYPICK_INTERPOLATENORMAL – Interpolate normal.

lplpPicked2Array
The address of a pointer to be initialized with a valid pointer to the **IDirect3DRMPicked2Array** interface. You then call the **IDirect3DRMPicked2Array::GetPick** method to retrieve a visual object, an **IDirect3DRMFrameArray** interface, and a **D3DRMPICKDESC2** structure. The array of frames is the path through the hierarchy leading to the visual object that intersected the ray. The **D3DRMPICKDESC2** structure contains the face and group identifiers, pick position, horizontal and vertical texture coordinates for the vertex, vertex normal, and color of the intersected objects.

If you specify D3DRMRAYPICK_ONLYBOUNDINGBOXES, the texture, normal and color data in the **D3DRMPICKDESC2** structure will be invalid.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

There are two kinds of flags: optimization flags and interpolation flags. Optimization flags allow you to limit the search and therefore make it faster. Interpolation flags specify what to interpolate if a primitive is hit. The three interpolation choices are color, normal, and texture coordinates.

The ray is specified in the reference frame coordinate space (pointed to by *lpRefFrame*). If the reference frame is NULL, the ray is specified in world coordinates.

IDirect3DRMFrame2::Save

Saves a Direct3DRMFrame2 object to the specified file.

```
HRESULT Save(
    LPCSTR lpFilename,
    D3DRMXOFFORMAT d3dFormat,
    D3DRMSAVEOPTIONS d3dSaveFlags
    );
```

Parameters

lpFilename
Address specifying the name of the created file. This file must have a .X file name extension.

d3dFormat
The D3DRMXOF_TEXT value from the **D3DRMXOFFORMAT** enumerated type.

d3dSaveFlags
Value of the **D3DRMSAVEOPTIONS** type describing the save options.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::SetAxes

Sets the vectors that define a coordinate space by which the **IDirect3DRMFrame2::SetOrientation** vectors are transformed.

```
HRESULT SetAxes(
    D3DVALUE dx,
    D3DVALUE dy,
    D3DVALUE dz,
    D3DVALUE ux,
    D3DVALUE uy,
    D3DVALUE uz
    );
```

Parameters

dx, dy, dz
The z-axis for the frame. Default is (0,0,1).

ux, uy, uz
The y-axis for the frame. Default is (0,1,0).

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method helps support both left-handed and right-handed coordinate systems. This method also allows you to specify that the negative z-axis represents the front of the object.

The **IDirect3DRMFrame2::SetOrientation** direction (*rvDx*, *rvDy*, *rvDz*) and up (*rvUx*, *rvUy*, *rvUz*) vectors are transformed according to the value of the **SetAxes** vectors.

The axes are inherited by child frames as specified in **IDirect3DRMFrame2::SetInheritAxes**.

See Also **IDirect3DRMFrame2::GetAxes**, **IDirect3DRMFrame2::GetInheritAxes**, **IDirect3DRMFrame2::SetInheritAxes**

IDirect3DRMFrame2::SetBox

Sets the box to be used in bounding box testing. In order for the bounding box to be valid, its minimum x must be less than or equal to its maximum x, minimum y must be less than or equal to its maximum y, and minimum z must be less than or equal to its maximum z.

HRESULT SetBox(
D3DRMBOX * *lpD3DRMBox*
);

Parameters

lpD3DRMBox
Address of a **D3DRMBOX** structure contains the bounding box coordinates.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method supports a bounding box on a frame for hierarchical culling. For a bounding box to be enabled, the **IDirect3DRMFrame2::SetBoxEnable** method must be called to set the enable flag to TRUE. By default, the box enable flag is FALSE.

See Also **IDirect3DRMFrame2::GetBox**, **IDirect3DRMFrame2::SetBoxEnable**, **IDirect3DRMFrame2::GetBoxEnable**

IDirect3DRMFrame2::SetBoxEnable

Enables or disables bounding box testing for this Direct3DRMFrame2 object. Bounding box testing cannot be enabled unless a valid bounding box has already been set on the frame.

HRESULT SetBoxEnable(
BOOL *bEnableFlag*
);

Parameters

bEnableFlag
TRUE to enable a bounding box. FALSE if a bounding box is not enabled. Default is FALSE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

For a bounding box to be enabled, this method must be called to set the enable flag to TRUE. By default, the box enable flag is FALSE.

Bounding box testing is performed as follows: at render time the bounding box is transformed into model space and checked for intersection with the viewing frustum. If all of the box is outside of the viewing frustum, none of the visuals in the frame or in any child frames are rendered. Otherwise, rendering continues as normal.

Enabling bounding box testing with a box of {0,0,0,0} completely prevents a frame from being rendered.

See Also

IDirect3DRMFrame2::GetBoxEnable, **IDirect3DRMFrame2::GetBox**, **IDirect3DRMFrame2::SetBox**

IDirect3DRMFrame2::SetColor

Sets the color of the frame. This color is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME.

HRESULT SetColor(
D3DCOLOR *rcColor*
);

Parameters

rcColor
New color for the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a color key to a **DIRECT3DRMFRAMEINTERPOLATOR** object.

See Also **IDirect3DRMFrame2::GetColor**, **IDirect3DRMFrame2::SetMaterialMode**

IDirect3DRMFrame2::SetColorRGB

Sets the color of the frame. This color is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME.

HRESULT SetColorRGB(
D3DVALUE *rvRed*,
D3DVALUE *rvGreen*,
D3DVALUE *rvBlue*
);

Parameters

rvRed, rvGreen, and *rvBlue*
New color for the frame. Each component of the color should be in the range 0 to 1.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add an RGB color key to a DIRECT3DRMFRAMEINTERPOLATOR object.

See Also **IDirect3DRMFrame2::SetMaterialMode**

IDirect3DRMFrame2::SetInheritAxes

Specifies whether the axes for the frame are inherited from the parent frame.

HRESULT SetInheritAxes(
BOOL *inherit_from_parent*
);

Parameters

inherit_from_parent
Flag indicating whether the frame should inherit axes from its parent. If TRUE, the frame inherits axes (the default). If FALSE, the frame does not inherit axes.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks By default, the axes are inherited from the parent. If a frame is set to inherit from the parent and there is no parent, the frame acts as if it inherits from a parent with the default axes (direction=(0,0,1)and up=(0,1,0)). This method allows a single policy for axes to be set at the root of the hierarchy.

See Also **IDirect3DRMFrame2::GetInheritAxes**, **IDirect3DRMFrame2::GetAxes**, **IDirect3DRMFrame2::SetAxes**

IDirect3DRMFrame2::SetMaterial

Sets the material of the Direct3DRMFrame2 object.

HRESULT SetMaterial(
LPDIRECT3DRMMATERIAL **lplpMaterial*
);

Parameters *lplpMaterial*
Address of the Direct3DRMMaterial object that will be applied to the frame.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::GetMaterial**

IDirect3DRMFrame2::SetMaterialMode

Sets the material mode for a frame. The material mode determines the source of material information for visuals rendered with the frame.

HRESULT SetMaterialMode(
D3DRMMATERIALMODE *rmmMode*
);

Parameters *rmmMode*
One of the members of the **D3DRMMATERIALMODE** enumerated type.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::GetMaterialMode**

IDirect3DRMFrame2::SetOrientation

Aligns a frame so that its z-direction points along the direction vector [*rvDx, rvDy, rvDz*] and its y-direction aligns with the vector [*rvUx, rvUy, rvUz*].

HRESULT SetOrientation(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvDx*,
D3DVALUE *rvDy*,
D3DVALUE *rvDz*,
D3DVALUE *rvUx*,
D3DVALUE *rvUy*,
D3DVALUE *rvUz*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvDx, rvDy, and *rvDz*
New z-axis for the frame.

rvUx, rvUy, and *rvUz*
New y-axis for the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The default orientation of a frame has a direction vector of [0, 0, 1] and an up vector of [0, 1, 0].

If [*rvUx, rvUy, rvUz*] is parallel to [*rvDx, rvDy, rvDz*], the D3DRMERR_BADVALUE error value is returned; otherwise, the [*rvUx, rvUy, rvUz*] vector passed is projected onto the plane that is perpendicular to [*rvDx, rvDy, rvDz*].

This method is also used to add an orientation key to a **Direct3DRMFrameInterpolator** object.

See Also

IDirect3DRMFrame2::GetOrientation

IDirect3DRMFrame2::SetPosition

Sets the position of a frame relative to the frame of reference. It places the frame a distance of [*rvX, rvY, rvZ*] from the reference. When a child frame is created within a parent, it is placed at [0, 0, 0] in the parent frame.

HRESULT SetPosition(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
New position for the frame.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a position key to a **Direct3DRMFrameInterpolator** object.

See Also

IDirect3DRMFrame2::GetPosition

IDirect3DRMFrame2::SetQuaternion

Sets a frame's orientation relative to a reference frame using a unit quaternion.

HRESULT SetQuaternion(
LPDIRECT3DRMFRAME2 *lpRef*,
D3DRMQUATERNION **quat*
)

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame2 object to be used as the reference.

quat
A **D3DRMQUATERNION** structure that holds the unit quaternion.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks A quaternion is a four-valued vector that can be used to represent any rotation, and that has properties that are useful when interpolating between orientations. A quaternion is a unit quaternion if s**2 + x**2 + y**2 + z**2 = 1

The function **D3DRMQuaternionFromRotation** can be used to generate unit quaternions from arbitrary rotation values.

The **SetQuaternion** method is supported by FrameInterpolators. See *IDirect3DRMInterpolator Interface* for more information about interpolators.

IDirect3DRMFrame2::SetRotation

Sets a frame rotating by the given angle around the given vector at each call to the **IDirect3DRM::Tick** or **IDirect3DRMFrame2::Move** method. The direction vector [*rvX, rvY, rvZ*] is defined in the reference frame.

HRESULT SetRotation(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*,
D3DVALUE *rvTheta*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
Vector about which rotation occurs.

rvTheta
Rotation angle, in radians.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The specified rotation changes the matrix with every render tick, unlike the **IDirect3DRMFrame2::AddRotation** method, which changes the objects in the frame only once.

See Also **IDirect3DRMFrame2::AddRotation**, **IDirect3DRMFrame2::GetRotation**

IDirect3DRMFrame2::SetSceneBackground

Sets the background color of a scene.

HRESULT SetSceneBackground(
D3DCOLOR *rcColor*
);

Parameters

rcColor
New color for the background.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a background color key to a Direct3DRMFrameInterpolator object.

IDirect3DRMFrame2::SetSceneBackgroundDepth

Specifies a background-depth buffer for a scene.

HRESULT SetSceneBackgroundDepth(
LPDIRECTDRAWSURFACE *lpImage*
);

Parameters

lpImage
Address of a DirectDraw surface that will store the new background depth for the scene.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The image must have a depth of 16. If the image and viewport sizes are different, the image is scaled first. For best performance when animating the background-depth buffer, the image should be the same size as the viewport. This enables the depth buffer to be updated directly from the image memory without incurring extra overhead.

See Also

IDirect3DRMFrame2::GetSceneBackgroundDepth

IDirect3DRMFrame2::SetSceneBackgroundImage

Specifies a background image for a scene.

HRESULT SetSceneBackgroundImage(
LPDIRECT3DRMTEXTURE *lpTexture*
);

Parameters

lpTexture
Address of a Direct3DRMTexture object that will contain the new background scene.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

If the image is a different size or color depth than the viewport, the image will first be scaled or converted to the correct depth. For best performance when animating the background, the image should be the same size and color depth. This enables the background to be rendered directly from the image memory without incurring any extra overhead.

IDirect3DRMFrame2::SetSceneBackgroundRGB

Sets the background color of a scene.

HRESULT SetSceneBackgroundRGB(
D3DVALUE *rvRed*,
D3DVALUE *rvGreen*,
D3DVALUE *rvBlue*
);

Parameters

rvRed, rvGreen, and *rvBlue*
New color for the background.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a background RGB color key to a Direct3DRMFrameInterpolator object.

IDirect3DRMFrame2::SetSceneFogColor

Sets the fog color of a scene.

HRESULT SetSceneFogColor(
D3DCOLOR *rcColor*
);

Parameters *rcColor*
New color for the fog.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a fog color key to a Direct3DRMRFrameInterpolator object.

IDirect3DRMFrame2::SetSceneFogEnable

Sets the fog enable state.

HRESULT SetSceneFogEnable(
BOOL *bEnable*
);

Parameters *bEnable*
New fog enable state.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMFrame2::SetSceneFogMode

Sets the fog mode.

HRESULT SetSceneFogMode(
D3DRMFOGMODE *rfMode*
);

Parameters

rfMode
One of the members of the **D3DRMFOGMODE** enumerated type, specifying the new fog mode.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::SetSceneFogParams

IDirect3DRMFrame2::SetSceneFogParams

Sets the current fog parameters for this scene.

HRESULT SetSceneFogParams(
D3DVALUE *rvStart*,
D3DVALUE *rvEnd*,
D3DVALUE *rvDensity*
);

Parameters

rvStart and *rvEnd*
Fog start and end points for linear fog mode. These settings determine the distance from the camera at which fog effects first become visible and the distance at which fog reaches its maximum density.

rvDensity
Fog density for the exponential fog modes. This value should be in the range 0 through 1.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a fog parameters key to a Direct3DRMRFrameInterpolator object.

See Also

D3DRMFOGMODE, **IDirect3DRMFrame2::SetSceneFogMode**

IDirect3DRMFrame2::SetSortMode

Sets the sorting mode used to process child frames. You can use this method to change the properties of hidden-surface-removal algorithms.

HRESULT SetSortMode(
D3DRMSORTMODE *d3drmSM*
);

Parameters

d3drmSM
One of the members of the **D3DRMSORTMODE** enumerated type, specifying the sorting mode. The default value is D3DRMSORT_FROMPARENT.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::GetSortMode

IDirect3DRMFrame2::SetTexture

Sets the texture of the frame.

HRESULT SetTexture(
LPDIRECT3DRMTEXTURE *lpD3DRMTexture*
);

Parameters

lpD3DRMTexture
Address of a variable that represents the Direct3DRMTexture object to be used.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The texture is used for meshes in the frame when the **D3DRMMATERIALMODE** enumerated type is D3DRMMATERIAL_FROMFRAME. To disable the frame's texture, use a NULL texture.

See Also

IDirect3DRMFrame2::GetTexture, **IDirect3DRMFrame2::SetMaterialMode**

IDirect3DRMFrame2::SetTextureTopology

Defines the topological properties of the texture coordinates across objects in the frame.

HRESULT SetTextureTopology(
BOOL *bWrap_u*,
BOOL *bWrap_v*
);

Parameters

bWrap_u and *bWrap_v*
Variables that are set to TRUE to map the texture in the u- and v-directions, respectively.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMFrame2::GetTextureTopology

IDirect3DRMFrame2::SetVelocity

Sets the velocity of the given frame relative to the reference frame. The frame will be moved by the vector [*rvX, rvY, rvZ*] with respect to the reference frame at each successive call to the **IDirect3DRM::Tick** or **IDirect3DRMFrame2::Move** method.

HRESULT SetVelocity(
LPDIRECT3DRMFRAME *lpRef*,
D3DVALUE *rvX*,
D3DVALUE *rvY*,
D3DVALUE *rvZ*,
BOOL *fRotVel*
);

Parameters

lpRef
Address of a variable that represents the Direct3DRMFrame object to be used as the reference.

rvX, rvY, and *rvZ*
New velocity for the frame.

fRotVel
Flag specifying whether the rotational velocity of the object is taken into account when setting the linear velocity. If TRUE, the object's rotational velocity is included in the calculation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::GetVelocity**

IDirect3DRMFrame2::SetZbufferMode

Sets the z-buffer mode; that is, whether z-buffering is enabled or disabled.

HRESULT SetZbufferMode(
D3DRMZBUFFERMODE *d3drmZBM*
);

Parameters *d3drmZBM*
One of the members of the **D3DRMZBUFFERMODE** enumerated type, specifying the z-buffer mode. The default value is D3DRMZBUFFER_FROMPARENT.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::GetZbufferMode**

IDirect3DRMFrame2::Transform

Transforms the vector in the *lpd3dVSrc* parameter in model coordinates to world coordinates, returning the result in the *lpd3dVDst* parameter.

HRESULT Transform(
D3DVECTOR **lpd3dVDst*,
D3DVECTOR **lpd3dVSrc*
);

Parameters *lpd3dVDst*
Address of a **D3DVECTOR** structure that will be filled with the result of the transformation operation.

lpd3dVSrc
Address of a **D3DVECTOR** structure that is the source of the transformation operation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMFrame2::InverseTransform**, *3D Transformations*

IDirect3DRMInterpolator

Interpolators provide a way of storing actions and applying them to objects with automatic calculation of in-between values. With an interpolator you can blend colors, move objects smoothly between positions, morph meshes, and perform many other transformations.

The **IDirect3DRMInterpolator** interface is a superset of the **IDirect3DRMAnimation** interface that increases the kinds of object parameters you can animate. While **IDirect3DRMAnimation** allows animation of an object's position, size and orientation, **IDirect3DRMInterpolator** further enables animation of color, meshes, texture, and material.

For a conceptual overview, see *IDirect3DRMInterpolator Overview*.

In addition to the standard *IUnknown* and **IDirect3DRMObject** methods, **IDirect3DRMInterpolator** contains the following methods:

Attaching Objects	**AttachObject**
	DetachObject
	GetAttachedObjects
Interpolating	**GetIndex**
	Interpolate
	SetIndex

The **IDirect3DRMInterpolator** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, **IDirect3DRMInterpolator** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone
DeleteDestroyCallback
GetAppData
GetClassName
GetName
SetAppData
SetName

IDirect3DRMInterpolator::AttachObject

Connects an object to the interpolator.

HRESULT AttachObject(
LPDIRECT3DRMOBJECT *lpD3DRMObject*
)

Parameters

lpD3DRMObject
Address of the *Direct3DRMObject* object to be attached to the interpolator.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The attached object can be another interpolator or an object of type *x* where the interpolator is of type *x*Interpolator. For example, a Viewport can be attached to a ViewportInterpolator. The interpolator types are:

- FrameInterpolator
- LightInterpolator
- MaterialInterpolator
- MeshInterpolator
- TextureInterpolator
- ViewportInterpolator

IDirect3DRMInterpolator::DetachObject

Detaches an object from the interpolator.

HRESULT DetachObject(
LPDIRECT3DRMOBJECT *lpD3DRMObject*
)

Parameters

lpD3DRMObject
Address of the *Direct3DRMObject* object to be detached from the interpolator.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMInterpolator::GetAttachedObjects

Returns an array of objects currently attached to the interpolator.

HRESULT GetAttachedObjects(
LPDIRECT3DRMOBJECTARRAY *lpD3DRMObjectArray*
)

Parameters

lpD3DRMObjectArray
Address of an **IDirect3DRMObjectArray** object containing the *Direct3DRMObject* objects currently attached to the interpolator.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMInterpolator::GetIndex

Retrieves the interpolator's current internal index (time).

D3DVALUE GetIndex( )

Return Values Returns a **D3DVALUE** that contains the interpolator's current internal index.

Remarks Every key stored in an interpolator has an index value. When a key is recorded (by calling a method), the key is stamped with the current interpolator index value. The key's index value does not change after being stamped.

IDirect3DRMInterpolator::Interpolate

Generates a series of actions by interpolating between keys stored in the interpolator. The actions are then applied to the specified object. If no object is specified, the actions are applied to the currently attached objects.

HRESULT Interpolate(
D3DVALUE *d3dVal*,
LPDIRECT3DRMOBJECT *lpD3DRMObject*,
D3DRMINTERPOLATIONOPTIONS *d3drmInterpFlags*
)

Parameters

d3dVal
A D3DVALUE that contains the interpolator's current internal index.

lpD3DRMObject
Address of the *Direct3DRMObject* object which will be assigned interpolated values for all properties stored in the interpolator. Can be NULL, in which case the property values of all attached objects will be set to interpolated values.

d3drmInterpFlags
One of more flags that control the kind of interpolation done. Possible values are:

D3DRMINTERPOLATION_CLOSED

D3DRMINTERPOLATION_LINEAR

D3DRMINTERPOLATION_NEAREST

D3DRMINTERPOLATION_OPEN

D3DRMINTERPOLATION_SLERPNORMALS

D3DRMINTERPOLATION_SPLINE

D3DRMINTERPOLATION_VERTEXCOLOR

See *Interpolation Options* for a description of these options.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMInterpolator::SetIndex

Sets the interpolator's internal index (time) to the specified value. If other interpolators are attached to the interpolator, this method recursively synchronizes their indices to the same value.

HRESULT SetIndex(
D3DVALUE *d3dVal*
)

Parameters

d3dVal
The time to set for the interpolator's internal index.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Every key stored in an interpolator has an index value. When a key is recorded (by calling a method), the key is stamped with the current interpolator index value. The key's index value does not change after being stamped.

IDirect3DRMLight

Applications use the methods of the **IDirect3DRMLight** interface to interact with light objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMLight and IDirect3DRMLightArray Interfaces*.

The methods of the **IDirect3DRMLight** interface can be organized into the following groups:

Attenuation	**GetConstantAttenuation**
	GetLinearAttenuation
	GetQuadraticAttenuation
	SetConstantAttenuation
	SetLinearAttenuation
	SetQuadraticAttenuation
Color	**GetColor**
	SetColor
	SetColorRGB

Enable frames	**GetEnableFrame** **SetEnableFrame**
Light types	**GetType** **SetType**
Range	**GetRange** **SetRange**
Spotlight options	**GetPenumbra** **GetUmbra** **SetPenumbra** **SetUmbra**

The **IDirect3DRMLight** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMLight** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMLight object is obtained by calling the **IDirect3DRM::CreateLight** or **IDirect3DRM::CreateLightRGB** method.

IDirect3DRMLight::GetColor

Retrieves the color of the current Direct3DRMLight object.

D3DCOLOR GetColor( );

Return Values Returns the color.

See Also **IDirect3DRMLight::SetColor**

IDirect3DRMLight::GetConstantAttenuation

Retrieves the constant attenuation factor for the Direct3DRMLight object.

D3DVALUE GetConstantAttenuation( );

Return Values Returns the constant attenuation value.

Remarks Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

See Also **IDirect3DRMLight::SetConstantAttenuation**

IDirect3DRMLight::GetEnableFrame

Retrieves the enable frame for a light.

HRESULT GetEnableFrame(
LPDIRECT3DRMFRAME * *lplpEnableFrame*

```
);
```

Parameters

lplpEnableFrame
Address of a pointer that will contain the enable frame for the current Direct3DRMFrame object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMLight::SetEnableFrame

IDirect3DRMLight::GetLinearAttenuation

Retrieves the linear attenuation factor for a light.

```
D3DVALUE GetLinearAttenuation( );
```

Return Values

Returns the linear attenuation value.

Remarks

Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

See Also

IDirect3DRMLight::SetLinearAttenuation

IDirect3DRMLight::GetPenumbra

Retrieves the penumbra angle of a spotlight.

```
D3DVALUE GetPenumbra( );
```

Return Values

Returns the penumbra value.

See Also **IDirect3DRMLight::SetPenumbra**

IDirect3DRMLight::GetQuadraticAttenuation

Retrieves the quadratic attenuation factor for a light.

D3DVALUE GetQuadraticAttenuation( );

Return Values Returns the quadratic attenuation value.

Remarks Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

See Also **IDirect3DRMLight::SetQuadraticAttenuation**

IDirect3DRMLight::GetRange

Retrieves the range of the current Direct3DRMLight object.

D3DVALUE GetRange( );

Return Values Returns a value describing the range.

See Also **IDirect3DRMLight::SetRange**

IDirect3DRMLight::GetType

Retrieves the type of a given light.

D3DRMLIGHTTYPE GetType( );

Return Values Returns one of the members of the **D3DRMLIGHTTYPE** enumerated type.

See Also **IDirect3DRMLight::SetType**

IDirect3DRMLight::GetUmbra

Retrieves the umbra angle of the Direct3DRMLight object.

D3DVALUE GetUmbra( );

Return Values Returns the umbra angle.

See Also **IDirect3DRMLight::SetUmbra**

IDirect3DRMLight::SetColor

Sets the color of the given light.

HRESULT SetColor(
D3DCOLOR *rcColor*
);

Parameters *rcColor*
New color for the light.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a color key to a Direct3DRMLightInterpolator object.

See Also **IDirect3DRMLight::GetColor**

IDirect3DRMLight::SetColorRGB

Sets the color of the given light.

HRESULT SetColorRGB(
D3DVALUE *rvRed*,
D3DVALUE *rvGreen*,
D3DVALUE *rvBlue*
);

Parameters

rvRed, rvGreen, and *rvBlue*
New color for the light.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add an RGB color key to a Direct3DRMLightInterpolator object.

IDirect3DRMLight::SetConstantAttenuation

Sets the constant attenuation factor for a light.

HRESULT SetConstantAttenuation(
D3DVALUE *rvAtt*
);

Parameters

rvAtt
New constant attenuation factor.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

This method is also used to add a constant attenuation key to a Direct3DRMLightInterpolator object.

See Also **IDirect3DRMLight::GetConstantAttenuation**

IDirect3DRMLight::SetEnableFrame

Sets the enable frame for a light.

HRESULT SetEnableFrame(
LPDIRECT3DRMFRAME *lpEnableFrame*
);

Parameters *lpEnableFrame*
Address of the light's enable frame. Child frames of this frame are also enabled for this light source.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMLight::GetEnableFrame**

IDirect3DRMLight::SetLinearAttenuation

Sets the linear attenuation factor for a light.

HRESULT SetLinearAttenuation(
D3DVALUE *rvAtt*
);

Parameters *rvAtt*
New linear attenuation factor.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks
Remarks Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

This method is also used to add a linear attenuation key to a Direct3DRMLightInterpolator object.

See Also **IDirect3DRMLight::GetLinearAttenuation**

IDirect3DRMLight::SetPenumbra

Sets the angle of the penumbra cone.

HRESULT SetPenumbra(
D3DVALUE *rvAngle*
);

Parameters

rvAngle
New penumbra angle. This angle must be greater than or equal to the angle of the umbra. If you set the penumbra angle to less than the umbra angle, the umbra angle will be set equal to the penumbra angle. The default value is 0.5 radians.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a penumbra angle key to a Direct3DRMLightInterpolator object.

See Also **IDirect3DRMLight::GetPenumbra**

IDirect3DRMLight::SetQuadraticAttenuation

Sets the quadratic attenuation factor for a light.

HRESULT SetQuadraticAttenuation(
D3DVALUE *rvAtt*

);

Parameters

rvAtt
New quadratic attenuation factor.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks
Remarks

Lights that have a location (are not infinitely far away) can have attenuation factors to calculate the attenuation of the light based on distance from the light.

The formula for the total attenuation factor is:

1 / [*constant_attenuation_factor* + distance * *linear_attenuation_factor* + (distance**2) * *quadratic_attenuation_factor*]

The total attenuation factor cannot be greater than 1. When attenuation factors are not provided, the default values are 1.0 for the constant attenuation factor, 0.0 for the linear attenuation factor, and 0.0 for the quadratic attenuation factor.

This method is also used to add a quadratic attenuation key to a Direct3DRMLightInterpolator object.

See Also

IDirect3DRMLight::GetQuadraticAttenuation

IDirect3DRMLight::SetRange

Sets the range of a spot light. The light affects objects that are within the range only.

HRESULT SetRange(
D3DVALUE *rvRange*
);

Parameters

rvRange
New range. The default value is 256.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The range property is for spotlights only. This method is also used to add a range key to a Direct3DRMLightInterpolator object.

See Also

IDirect3DRMLight::GetRange

IDirect3DRMLight::SetType

Changes the light's type.

HRESULT SetType(
D3DRMLIGHTTYPE *d3drmtType*
);

Parameters

d3drmtType
New light type specified by one of the members of the **D3DRMLIGHTTYPE** enumerated type.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMLight::GetType

IDirect3DRMLight::SetUmbra

Sets the angle of the umbra cone.

HRESULT SetUmbra(
D3DVALUE *rvAngle*
);

Parameters

rvAngle
New umbra angle. This angle must be less than or equal to the angle of the penumbra. If you set the umbra angle to greater than the penumbra angle, the penumbra angle will be set equal to the umbra angle. The default value is 0.4 radians.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add an umbra angle key to a Direct3DRMLightInterpolator object.

See Also

IDirect3DRMLight::GetUmbra

IDirect3DRMMaterial

Applications use the methods of the **IDirect3DRMMaterial** interface to interact with material objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMMaterial Interface*.

The methods of the **IDirect3DRMMaterial** interface can be organized into the following groups:

Emission	**GetEmissive**
	SetEmissive
Power for	**GetPower**
specular exponent	**SetPower**
Specular	**GetSpecular**
	SetSpecular

The **IDirect3DRMMaterial** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMMaterial** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMMaterial object is obtained by calling the **IDirect3DRM::CreateMaterial** method.

IDirect3DRMMaterial::GetEmissive

Retrieves the setting for the emissive property of a material. The setting of this property is the color and intensity of the light the object emits.

HRESULT GetEmissive(
D3DVALUE **lpr*,
D3DVALUE **lpg*,
D3DVALUE **lpb*
);

Parameters

lpr, *lpg*, and *lpb*
Addresses that will contain the red, green, and blue components of the emissive color when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMaterial::SetEmissive

IDirect3DRMMaterial::GetPower

Retrieves the power used for the specular exponent in the given material.

D3DVALUE GetPower( );

Return Values

Returns the value specifying the power of the specular exponent.

See Also

IDirect3DRMMaterial::SetPower

IDirect3DRMMaterial::GetSpecular

Retrieves the color of the specular highlights of a material.

HRESULT GetSpecular(
D3DVALUE **lpr*,
D3DVALUE **lpg*,
D3DVALUE **lpb*

);

Parameters *lpr*, *lpg*, and *lpb*
Addresses that will contain the red, green, and blue components of the color of the specular highlights when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMaterial::SetSpecular**

IDirect3DRMMaterial::SetEmissive

Sets the emissive property of a material.

HRESULT SetEmissive(
D3DVALUE *r*,
D3DVALUE *g*,
D3DVALUE *b*
);

Parameters *r*, *g*, and *b*
Red, green, and blue components of the emissive color.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add an emissive property key to a Direct3DRMMaterialInterpolator object.

See Also **IDirect3DRMMaterial::GetEmissive**

IDirect3DRMMaterial::SetPower

Sets the power used for the specular exponent in a material.

HRESULT SetPower(
D3DVALUE *rvPower*
);

Parameters *rvPower*
New specular exponent.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a spectral power key to a Direct3DRMMaterialInterpolator object.

See Also **IDirect3DRMMaterial::GetPower**

IDirect3DRMMaterial::SetSpecular

Sets the color of the specular highlights for a material.

```
HRESULT SetSpecular(
    D3DVALUE r,
    D3DVALUE g,
    D3DVALUE b
    );
```

Parameters *r*, *g*, and *b*
Red, green, and blue components of the color of the specular highlights.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a spectral color key to a Direct3DRMMaterialInterpolator object.

See Also **IDirect3DRMMaterial::GetSpecular**

IDirect3DRMMesh

Applications use the methods of the **IDirect3DRMMesh** interface to interact with groups of meshes. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMMesh, IDirect3DRMMeshBuilder, and IDirect3DRMMeshBuilder2 Interfaces*.

The methods of the **IDirect3DRMMesh** interface can be organized into the following groups:

Color	**GetGroupColor**
	SetGroupColor

	SetGroupColorRGB
Creation and information	**AddGroup** **GetBox** **GetGroup** **GetGroupCount**
Materials	**GetGroupMaterial** **SetGroupMaterial**
Miscellaneous	**Scale** **Translate**
Rendering quality	**GetGroupQuality** **SetGroupQuality**
Texture mapping	**GetGroupMapping** **SetGroupMapping**
Textures	**GetGroupTexture** **SetGroupTexture**
Vertex positions	**GetVertices** **SetVertices**

The **IDirect3DRMMesh** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMMesh** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMMesh object is obtained by calling the **IDirect3DRM::CreateMesh** method.

IDirect3DRMMesh::AddGroup

Groups a collection of faces and retrieves an identifier for the group. The first group added to a mesh always has index 0. The index of each successive group increases by 1.

```
HRESULT AddGroup(
    unsigned vCount,
    unsigned fCount,
    unsigned vPerFace,
    unsigned *fData,
    D3DRMGROUPINDEX *returnId
    );
```

Parameters

vCount and *fCount*
Number of vertices and faces in the group.

vPerFace
Number of vertices per face in the group, if all faces have the same vertex count. If the group contains faces with varying vertex counts, this parameter should be zero.

fData
Address of face data. If the *vPerFace* parameter specifies a value, this data is simply a list of indices into the group's vertex array. If *vPerFace* is zero, the vertex indices should be preceded by an integer giving the number of vertices in that face. For example, if *vPerFace* is zero and the group is made up of triangular and quadrilateral faces, the data might be in the following form: 3 *index index index* 4 *index index index index* 3 *index index index ...*

returnId
Address of a variable that will identify the group when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks A newly added group has the following default properties:

- White
- No texture
- No specular reflection
- Position, normal, and color of each vertex in the vertex array equal to zero

To set the positions of the vertices, use the **IDirect3DRMMesh::SetVertices** method.

IDirect3DRMMesh::GetBox

Retrieves the bounding box containing a Direct3DRMMesh object. The bounding box gives the minimum and maximum model coordinates in each dimension.

HRESULT GetBox(
D3DRMBOX * *lpD3DRMBox*
);

Parameters *lpD3DRMBox*
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMesh::GetGroup

Retrieves the data associated with a specified group.

HRESULT GetGroup(
D3DRMGROUPINDEX *id*,
unsigned **vCount*,
unsigned **fCount*,
unsigned **vPerFace*,
DWORD **fDataSize*,
unsigned **fData*

);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

vCount and *fCount*
Addresses of variables that will contain the number of vertices and the number of faces for the group when the method returns. These parameters can be NULL.

vPerFace
Address of a variable that will contain the number of vertices per face for the group when the method returns. This parameter can be NULL.

fDataSize
Address of a variable that specifies the number of unsigned elements in the buffer pointed to by the *fData* parameter. This parameter cannot be NULL.

fData
Address of a buffer that will contain the face data for the group when the method returns. The format of this data is the same as was specified in the call to the **IDirect3DRMMesh::AddGroup** method. If this parameter is NULL, the method returns the required size of the buffer in the *fDataSize* parameter.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMesh::GetGroupColor

Retrieves the color for a group.

D3DCOLOR GetGroupColor(
D3DRMGROUPINDEX *id*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

Return Values

Returns a D3DCOLOR variable specifying the color if successful, or zero otherwise.

See Also

IDirect3DRMMesh::SetGroupColor, **IDirect3DRMMesh::SetGroupColorRGB**

IDirect3DRMMesh::GetGroupCount

Retrieves the number of groups for a given Direct3DRMMesh object.

unsigned GetGroupCount( );

Return Values Returns the number of groups if successful, or zero otherwise.

IDirect3DRMMesh::GetGroupMapping

Returns a description of how textures are mapped to a group in a Direct3DRMMesh object.

D3DRMMAPPING GetGroupMapping(
D3DRMGROUPINDEX *id*
);

Parameters *id*
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

Return Values Returns one of the **D3DRMMAPPING** values describing how textures are mapped to a group, if successful. Returns zero otherwise.

See Also **IDirect3DRMMesh::SetGroupMapping**

IDirect3DRMMesh::GetGroupMaterial

Retrieves a pointer to the material associated with a group in a Direct3DRMMesh object.

HRESULT GetGroupMaterial(
D3DRMGROUPINDEX *id*,
LPDIRECT3DRMMATERIAL **returnPtr*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

returnPtr
Address of a pointer to a variable that will contain the *IDirect3DRMMaterial* interface for the group when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMesh::SetGroupMaterial

IDirect3DRMMesh::GetGroupQuality

Retrieves the rendering quality for a specified group in a Direct3DRMMesh object.

D3DRMRENDERQUALITY GetGroupQuality(
D3DRMGROUPINDEX *id*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

Return Values

Returns values from the enumerated types represented by **D3DRMRENDERQUALITY** if successful, or zero otherwise. These values include the shading, lighting, and fill modes for the object.

See Also

IDirect3DRMMesh::SetGroupQuality

IDirect3DRMMesh::GetGroupTexture

Retrieves an address of the texture associated with a group in a Direct3DRMMesh object.

HRESULT GetGroupTexture(
D3DRMGROUPINDEX *id*,
LPDIRECT3DRMTEXTURE **returnPtr*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

returnPtr
Address of a pointer to a variable that will contain the *IDirect3DRMTexture* interface for the group when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMesh::SetGroupTexture

IDirect3DRMMesh::GetVertices

Retrieves vertex information for a specified group in a Direct3DRMMesh object.

```
HRESULT GetVertices(
    D3DRMGROUPINDEX id,
    DWORD index,
    DWORD count,
    D3DRMVERTEX *returnPtr
    );
```

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

index
Index into the array of **D3DRMVERTEX** structures at which to begin returning vertex positions.

count
Number of **D3DRMVERTEX** structures (vertices) to retrieve following the index given in the *index* parameter. This parameter cannot be NULL. To retrieve the number of vertices in a group, call the **IDirect3DRMMesh::GetGroup** method.

returnPtr
Array of **D3DRMVERTEX** structures that will contain the vertex information (vertex positions, colors, texture coordinates, and so on) when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMesh::GetGroup, **IDirect3DRMMesh::SetVertices**

IDirect3DRMMesh::Scale

Scales a Direct3DRMMesh object by the given scaling factors, parallel to the x-, y-, and z-axes in model coordinates.

HRESULT Scale(
D3DVALUE *sx*,
D3DVALUE *sy*,
D3DVALUE *sz*
);

Parameters

sx, sy, and *sz*
Scaling factors that are applied along the x-, y-, and z-axes.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMesh::SetGroupColor

Sets the color of a group in a Direct3DRMMesh object.

HRESULT SetGroupColor(
D3DRMGROUPINDEX *id*,
D3DCOLOR *value*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

value
Color of the group.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a group color key to a Direct3DRMMeshInterpolator object.

See Also

IDirect3DRMMesh::GetGroupColor, **IDirect3DRMMesh::SetGroupColorRGB**

IDirect3DRMMesh::SetGroupColorRGB

Sets the color of a group in a Direct3DRMMesh object, using individual RGB values.

HRESULT SetGroupColorRGB(
D3DRMGROUPINDEX *id*,
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

red, *green*, and *blue*
Red, green, and blue components of the group color.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a group RGB key to a Direct3DRMMeshInterpolator object.

See Also

IDirect3DRMMesh::GetGroupColor, **IDirect3DRMMesh::SetGroupColor**

IDirect3DRMMesh::SetGroupMapping

Sets the mapping for a group in a Direct3DRMMesh object. The mapping controls how textures are mapped to a surface.

HRESULT SetGroupMapping(
D3DRMGROUPINDEX *id*,
D3DRMMAPPING *value*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

value
Value of the **D3DRMMAPPING** type describing the mapping for the group.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMesh::GetGroupMapping**

IDirect3DRMMesh::SetGroupMaterial

Sets the material associated with a group in a Direct3DRMMesh object.

HRESULT SetGroupMaterial(
D3DRMGROUPINDEX *id*,
LPDIRECT3DRMMATERIAL *value*
);

Parameters *id*
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

value
Address of the *IDirect3DRMMaterial* interface for the Direct3DRMMesh object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMesh::GetGroupMaterial**

IDirect3DRMMesh::SetGroupQuality

Sets the rendering quality for a specified group in a Direct3DRMMesh object.

HRESULT SetGroupQuality(
D3DRMGROUPINDEX *id*,
D3DRMRENDERQUALITY *value*
);

Parameters *id*
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

value
Values from the enumerated types represented by the **D3DRMRENDERQUALITY** type. These values include the shading, lighting, and fill modes for the object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMesh::GetGroupQuality**

IDirect3DRMMesh::SetGroupTexture

Sets the texture associated with a group in a Direct3DRMMesh object.

HRESULT SetGroupTexture(
D3DRMGROUPINDEX *id*,
LPDIRECT3DRMTEXTURE *value*
);

Parameters *id*
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

value
Address of the *IDirect3DRMTexture* interface for the Direct3DRMMesh object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMesh::GetGroupTexture**

IDirect3DRMMesh::SetVertices

Sets the vertex positions for a specified group in a Direct3DRMMesh object.

HRESULT SetVertices(
D3DRMGROUPINDEX *id*,
unsigned *index*,
unsigned *count*,
D3DRMVERTEX **values*
);

Parameters

id
Identifier of the group. This identifier must have been produced by using the **IDirect3DRMMesh::AddGroup** method.

index
Index of the first vertex in the mesh group to be modified.

count
Number of consecutive vertices to set. The *values* array must contain at least *count* elements.

values
Array of **D3DRMVERTEX** structures specifying the vertex positions to be set.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Vertices are local to the group. If an application needs to share vertices between two different groups (for example, if neighboring faces in a mesh are different colors), the vertices must be duplicated in both groups.

This method is also used to add a vertex position key to a Direct3DRMMeshInterpolator object.

See Also

IDirect3DRMMesh::GetVertices

IDirect3DRMMesh::Translate

Adds the specified offsets to the vertex positions of a Direct3DRMMesh object.

HRESULT Translate(
D3DVALUE *tx*,
D3DVALUE *ty*,
D3DVALUE *tz*
);

Parameters

tx, ty, and *tz*
Offsets that are added to the x-, y-, and z-coordinates respectively of each vertex position.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder

Applications use the methods of the **IDirect3DRMMeshBuilder** interface to interact with mesh objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMMesh, IDirect3DRMMeshBuilder, and IDirect3DRMMeshBuilder2 Interfaces*.

The methods of the **IDirect3DRMMeshBuilder** interface can be organized into the following groups:

Color	**GetColorSource**
	SetColor
	SetColorRGB
	SetColorSource
Creation and information	**GetBox**
Faces	**AddFace**
	AddFaces
	CreateFace
	GetFaceCount
	GetFaces
Loading	**Load**
Meshes	**AddMesh**
	CreateMesh
Miscellaneous	**AddFrame**
	AddMeshBuilder
	ReserveSpace
	Save
	Scale
	SetMaterial
	Translate
Normals	**AddNormal**
	GenerateNormals

	SetNormal
Perspective	**GetPerspective**
	SetPerspective
Rendering quality	**GetQuality**
	SetQuality
Textures	**GetTextureCoordinates**
	SetTexture
	SetTextureCoordinates
	SetTextureTopology
Vertices	**AddVertex**
	GetVertexColor
	GetVertexCount
	GetVertices
	SetVertex
	SetVertexColor
	SetVertexColorRGB

The **IDirect3DRMMeshBuilder** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMMeshBuilder** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMMeshBuilder object is obtained by calling the **IDirect3DRM::CreateMeshBuilder** method.

IDirect3DRMMeshBuilder::AddFace

Adds a face to a Direct3DRMMeshBuilder object.

HRESULT AddFace(
LPDIRECT3DRMFACE *lpD3DRMFace*
);

Parameters

lpD3DRMFace
Address of the face being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Any one face can exist in only one mesh at a time.

IDirect3DRMMeshBuilder::AddFaces

Adds faces to a Direct3DRMMeshBuilder object.

HRESULT AddFaces(
DWORD *dwVertexCount*,
D3DVECTOR * *lpD3DVertices*,
DWORD *normalCount*,
D3DVECTOR **lpNormals*,
DWORD **lpFaceData*,
LPDIRECT3DRMFACEARRAY* *lplpD3DRMFaceArray*
);

Parameters

dwVertexCount
Number of vertices.

lpD3DVertices
Base address of an array of **D3DVECTOR** structures that stores the vertex positions.

normalCount
Number of normals.

lpNormals
Base address of an array of **D3DVECTOR** structures that stores the normal positions.

lpFaceData
For each face, this parameter should contain a vertex count followed by the indices into the vertices array. If *normalCount* is not zero, this parameter should contain a vertex count followed by pairs of indices, with the first index of each pair indexing into the array of vertices, and the second indexing into the array of normals. The list of indices must terminate with a zero.

lplpD3DRMFaceArray
Address of a pointer to an *IDirect3DRMFaceArray* interface that will be filled with a pointer to the newly created faces.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::AddFrame

Adds the contents of a frame to a Direct3DRMMeshBuilder object.

HRESULT AddFrame(
LPDIRECT3DRMFRAME *lpD3DRMFrame*
);

Parameters *lpD3DRMFrame*
Address of the frame whose contents are being added.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The source frame is not modified or referenced by this operation.

IDirect3DRMMeshBuilder::AddMesh

Adds a mesh to a Direct3DRMMeshBuilder object.

HRESULT AddMesh(
LPDIRECT3DRMMESH *lpD3DRMMesh*

);

Parameters

lpD3DRMMesh
Address of the mesh being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::AddMeshBuilder

Adds the contents of a Direct3DRMMeshBuilder object to another Direct3DRMMeshBuilder object.

HRESULT AddMeshBuilder(
LPDIRECT3DRMMESHBUILDER *lpD3DRMMeshBuild*
);

Parameters

lpD3DRMMeshBuild
Address of the Direct3DRMMeshBuilder object whose contents are being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The source Direct3DRMMeshBuilder object is not modified or referenced by this operation.

IDirect3DRMMeshBuilder::AddNormal

Adds a normal to a Direct3DRMMeshBuilder object.

int AddNormal(
D3DVALUE *x*,
D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

x, y, and *z*
The x, y, and z components of the direction of the new normal.

Return Values

Returns the index of the normal.

IDirect3DRMMeshBuilder::AddVertex

Adds a vertex to a Direct3DRMMeshBuilder object.

int AddVertex(
D3DVALUE *x*,
D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

x, y, and *z*
The x, y, and z components of the position of the new vertex.

Return Values

Returns the index of the vertex.

IDirect3DRMMeshBuilder::CreateFace

Creates a new face with no vertices and adds it to a Direct3DRMMeshBuilder object.

HRESULT CreateFace(
LPDIRECT3DRMFACE* *lplpD3DRMFace*
);

Parameters

lplpD3DRMFace
Address of a pointer to an *IDirect3DRMFace* interface that will be filled with a pointer to the face that was created.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::CreateMesh

Creates a new mesh from a Direct3DRMMeshBuilder object.

HRESULT CreateMesh(
LPDIRECT3DRMMESH* *lplpD3DRMMesh*
);

Parameters

lplpD3DRMMesh
Address that will be filled with a pointer to an *IDirect3DRMMesh* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::GenerateNormals

Processes the Direct3DRMMeshBuilder object and generates vertex normals that are the average of each vertex's adjoining face normals.

HRESULT GenerateNormals( );

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Averaging the normals of back-to-back faces produces a zero normal.

IDirect3DRMMeshBuilder::GetBox

Retrieves the bounding box containing a Direct3DRMMeshBuilder object. The bounding box gives the minimum and maximum model coordinates in each dimension.

HRESULT GetBox(
D3DRMBOX **lpD3DRMBox*
);

Parameters

lpD3DRMBox
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::GetColorSource

Retrieves the color source of a Direct3DRMMeshBuilder object. The color source can be either a face or a vertex.

D3DRMCOLORSOURCE GetColorSource( );

Return Values Returns a member of the **D3DRMCOLORSOURCE** enumerated type.

See Also **IDirect3DRMMeshBuilder::SetColorSource**

IDirect3DRMMeshBuilder::GetFaceCount

Retrieves the number of faces in a Direct3DRMMeshBuilder object.

int GetFaceCount( );

Return Values Returns the number of faces.

IDirect3DRMMeshBuilder::GetFaces

Retrieves the faces of a Direct3DRMMeshBuilder object.

HRESULT GetFaces(
LPDIRECT3DRMFACEARRAY* *lplpD3DRMFaceArray*
);

Parameters *lplpD3DRMFaceArray*
Address of a pointer to an *IDirect3DRMFaceArray* interface that is filled with an address of the faces.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::GetPerspective

Determines whether perspective correction is on for a Direct3DRMMeshBuilder object.

BOOL GetPerspective( );

Return Values Returns TRUE if perspective correction is on, or FALSE otherwise.

IDirect3DRMMeshBuilder::GetQuality

Retrieves the rendering quality of a Direct3DRMMeshBuilder object.

D3DRMRENDERQUALITY GetQuality( );

Return Values Returns a member of the **D3DRMRENDERQUALITY** enumerated type that specifies the rendering quality of the mesh.

See Also **IDirect3DRMMeshBuilder::SetQuality**

IDirect3DRMMeshBuilder::GetTextureCoordinates

Retrieves the texture coordinates of a specified vertex in a Direct3DRMMeshBuilder object.

HRESULT GetTextureCoordinates(
DWORD *index*,
D3DVALUE **lpU*,
D3DVALUE **lpV*
);

Parameters

index
Index of the vertex.

lpU and *lpV*
Addresses of variables that will be filled with the texture coordinates of the vertex when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMeshBuilder::SetTextureCoordinates**

IDirect3DRMMeshBuilder::GetVertexColor

Retrieves the color of a specified vertex in a Direct3DRMMeshBuilder object.

D3DCOLOR GetVertexColor(
DWORD *index*
);

Parameters *index*
Index of the vertex.

Return Values Returns the color.

See Also **IDirect3DRMMeshBuilder::SetVertexColor**

IDirect3DRMMeshBuilder::GetVertexCount

Retrieves the number of vertices in a Direct3DRMMeshBuilder object.

int GetVertexCount( );

Return Values Returns the number of vertices.

IDirect3DRMMeshBuilder::GetVertices

Retrieves the vertices, normals, and face data for a Direct3DRMMeshBuilder object.

HRESULT GetVertices(
DWORD **vcount*,
D3DVECTOR **vertices*,
DWORD **ncount*,
D3DVECTOR **normals*,
DWORD **face_data_size*,
DWORD **face_data*
);

Parameters

vcount
Address of a variable that will contain the number of vertices.

vertices
Address of an array of **D3DVECTOR** structures that will contain the vertices for the Direct3DRMMeshBuilder object. If this parameter is NULL, the method returns the number of vertices in the *vcount* parameter.

ncount
Address of a variable that will contain the number of normals.

normals
Address of an array of **D3DVECTOR** structures that will contain the normals for the Direct3DRMMeshBuilder object. If this parameter is NULL, the method returns the number of normals in the *ncount* parameter.

face_data_size
Address of a variable that specifies the size of the buffer pointed to by the *face_data* parameter. The size is given in units of **DWORD** values. This parameter cannot be NULL.

face_data
Address of the face data for the Direct3DRMMeshBuilder object. This data is in the same format as specified in the **IDirect3DRMMeshBuilder::AddFaces** method except that it is null-terminated. If this parameter is NULL, the method returns the required size of the face-data buffer in the *face_data_size* parameter.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::Load

Loads a Direct3DRMMeshBuilder object.

HRESULT Load(
LPVOID *lpvObjSource*,
LPVOID *lpvObjID*,
D3DRMLOADOPTIONS *d3drmLOFlags*,

D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc*,
LPVOID *lpvArg*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by an object that require special formatting. This parameter can be NULL.

lpvArg
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

By default, this method loads the first mesh from the source specified in the *lpvObjSource* parameter.

IDirect3DRMMeshBuilder::ReserveSpace

Reserves space within a Direct3DRMMeshBuilder object for the specified number of vertices, normals, and faces. This allows the system to use memory more efficiently.

HRESULT ReserveSpace(
DWORD *vertexCount*,
DWORD *normalCount*,
DWORD *faceCount*
);

Parameters

vertexCount, normalCount, and *faceCount*
Number of vertices, normals, and faces to allocate space for.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::Save

Saves a Direct3DRMMeshBuilder object.

```
HRESULT Save(
    const char * lpFilename,
    D3DRMXOFFORMAT d3drmXOFFormat,
    D3DRMSAVEOPTIONS d3drmSOContents
    );
```

Parameters

lpFilename
Address specifying the name of the created file. This file must have a .X file name extension.

d3drmXOFFormat
The D3DRMXOF_TEXT value from the **D3DRMXOFFORMAT** enumerated type.

d3drmSOContents
Value of the **D3DRMSAVEOPTIONS** type describing the save options.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::Scale

Scales a Direct3DRMMeshBuilder object by the given scaling factors, parallel to the x-, y-, and z-axes in model coordinates.

```
HRESULT Scale(
    D3DVALUE sx,
    D3DVALUE sy,
    D3DVALUE sz
    );
```

Parameters *sx, sy,* and *sz*
Scaling factors that are applied along the x-, y-, and z-axes.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetColor

Sets all the faces of a Direct3DRMMeshBuilder object to a given color.

HRESULT SetColor(
D3DCOLOR *color*
);

Parameters *color*
Color of the faces.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetColorRGB

Sets all the faces of a Direct3DRMMeshBuilder object to a given color.

HRESULT SetColorRGB(
D3DVALUE *red***,**
D3DVALUE *green***,**
D3DVALUE *blue*
);

Parameters *red, green,* and *blue*
Red, green, and blue components of the color.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetColorSource

Sets the color source of a Direct3DRMMeshBuilder object.

HRESULT SetColorSource(
D3DRMCOLORSOURCE *source*
);

Parameters

source
Member of the **D3DRMCOLORSOURCE** enumerated type that specifies the new color source to use.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMeshBuilder::GetColorSource

IDirect3DRMMeshBuilder::SetMaterial

Sets the material of all the faces of a Direct3DRMMeshBuilder object.

HRESULT SetMaterial(
LPDIRECT3DRMMATERIAL *lpIDirect3DRMmaterial*
);

Parameters

lpIDirect3DRMmaterial
Address of *IDirect3DRMMaterial* interface for the Direct3DRMMeshBuilder object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetNormal

Sets the normal vector of a specified vertex in a Direct3DRMMeshBuilder object.

HRESULT SetNormal(
DWORD *index*,
D3DVALUE *x*,

D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

index
Index of the normal to be set.

x, y, and *z*
The x, y, and z components of the vector to assign to the specified normal.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetPerspective

Enables or disables perspective-correct texture-mapping for a Direct3DRMMeshBuilder object.

HRESULT SetPerspective(
BOOL *perspective*
);

Parameters

perspective
Specify TRUE if the mesh should be texture-mapped with perspective correction, or FALSE otherwise.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetQuality

Sets the rendering quality of a Direct3DRMMeshBuilder object.

HRESULT SetQuality(
D3DRMRENDERQUALITY *quality*
);

Parameters

quality
Member of the **D3DRMRENDERQUALITY** enumerated type that specifies the new rendering quality to use.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An object's quality has three components: shade mode (flat or Gouraud, Phong is not yet implemented and will default to Gouraud shading), lighting type (on or off), and fill mode (point, wire-frame or solid).

You can set the quality of a device with **IDirect3DRMDevice::SetQuality**. By default it is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMMeshBuilder object with the **SetQuality** method. By default, a Direct3DRMMeshBuilder object's quality is D3DRMRENDER_GOURAUD (Gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

See Also

IDirect3DRMMeshBuilder::GetQuality

IDirect3DRMMeshBuilder::SetTexture

Sets the texture of all the faces of a Direct3DRMMeshBuilder object.

HRESULT SetTexture(
LPDIRECT3DRMTEXTURE *lpD3DRMTexture*
);

Parameters

lpD3DRMTexture
Address of the required Direct3DRMTexture object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetTextureCoordinates

Sets the texture coordinates of a specified vertex in a Direct3DRMMeshBuilder object.

HRESULT SetTextureCoordinates(
DWORD *index*,
D3DVALUE *u*,
D3DVALUE *v*
);

Parameters

index
Index of the vertex to be set.

u and *v*
Texture coordinates to assign to the specified mesh vertex.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMeshBuilder::GetTextureCoordinates**

IDirect3DRMMeshBuilder::SetTextureTopology

Sets the texture topology of a Direct3DRMMeshBuilder object.

HRESULT SetTextureTopology(
BOOL *cylU*,
BOOL *cylV*
);

Parameters

cylU and *cylV*
Specify TRUE for either or both of these parameters if you want the texture to have a cylindrical topology in the u and v dimensions respectively; otherwise, specify FALSE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetVertex

Sets the position of a specified vertex in a Direct3DRMMeshBuilder object.

```
HRESULT SetVertex(
    DWORD index,
    D3DVALUE x,
    D3DVALUE y,
    D3DVALUE z
    );
```

Parameters

index
Index of the vertex to be set.

x, y, and *z*
The x, y, and z components of the position to assign to the specified vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::SetVertexColor

Sets the color of a specified vertex in a Direct3DRMMeshBuilder object.

```
HRESULT SetVertexColor(
    DWORD index,
    D3DCOLOR color
    );
```

Parameters

index
Index of the vertex to be set.

color
Color to assign to the specified vertex.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMeshBuilder::GetVertexColor**

IDirect3DRMMeshBuilder::SetVertexColorRGB

Sets the color of a specified vertex in a Direct3DRMMeshBuilder object.

HRESULT SetVertexColorRGB(
DWORD *index*,
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*
);

Parameters *index*
Index of the vertex to be set.

red, green, and *blue*
Red, green, and blue components of the color to assign to the specified vertex.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder::Translate

Adds the specified offsets to the vertex positions of a Direct3DRMMeshBuilder object.

HRESULT Translate(
D3DVALUE *tx*,
D3DVALUE *ty*,
D3DVALUE *tz*
);

Parameters *tx, ty,* and *tz*
Offsets that are added to the x-, y-, and z-coordinates respectively of each vertex position.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2

Applications use the methods of the **IDirect3DRMMeshBuilder2** interface to interact with mesh objects. **IDirect3DRMMeshBuilder2** has the same functionality as **IDirect3DRMMeshBuilder** and in addition has extended the **IDirect3DRMMeshBuilder2::GenerateNormals2** method to give greater control over the mesh normals generated. **IDirect3DRMMeshBuilder2** also allows you to retrieve a single mesh face with **IDirect3DRMMeshBuilder2::GetFace**.

This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMMesh, IDirect3DRMMeshBuilder, and IDirect3DRMMeshBuilder2 Interfaces*.

The methods of the **IDirect3DRMMeshBuilder2** interface can be organized into the following groups:

Color	**GetColorSource**
	SetColor
	SetColorRGB
	SetColorSource
Creation	**GetBox**
Faces	**AddFace**
	AddFaces
	CreateFace
	GetFaceCount
	GetFace
	GetFaces
Loading	**Load**
Meshes	**AddMesh**
	CreateMesh
Miscellaneous	**AddFrame**
	AddMeshBuilder
	ReserveSpace

	Save
	Scale
	SetMaterial
	Translate
Normals	**AddNormal**
	GenerateNormals2
	SetNormal
Perspective	**GetPerspective**
	SetPerspective
Rendering quality	**GetQuality**
	SetQuality
Textures	**GetTextureCoordinates**
	SetTexture
	SetTextureCoordinates
	SetTextureTopology
Vertices	**AddVertex**
	GetVertexColor
	GetVertexCount
	GetVertices
	SetVertex
	SetVertexColor
	SetVertexColorRGB

The **IDirect3DRMMeshBuilder2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMMeshBuilder2** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMMeshBuilder2 object is obtained by calling the **IDirect3DRM::CreateMeshBuilder** method.

IDirect3DRMMeshBuilder2::AddFace

Adds a face to a Direct3DRMMeshBuilder2 object.

HRESULT AddFace(
LPDIRECT3DRMFACE *lpD3DRMFace*
);

Parameters

lpD3DRMFace
Address of the face being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Any one face can exist in only one mesh at a time.

IDirect3DRMMeshBuilder2::AddFaces

Adds faces to a Direct3DRMMeshBuilder2 object.

HRESULT AddFaces(
DWORD *dwVertexCount*,
D3DVECTOR * *lpD3DVertices*,
DWORD *normalCount*,
D3DVECTOR **lpNormals*,
DWORD **lpFaceData*,

LPDIRECT3DRMFACEARRAY* *lplpD3DRMFaceArray*
);

Parameters

dwVertexCount
Number of vertices.

lpD3DVertices
Base address of an array of **D3DVECTOR** structures that stores the vertex positions.

normalCount
Number of normals.

lpNormals
Base address of an array of **D3DVECTOR** structures that stores the normal positions.

lpFaceData
For each face, this parameter should contain a vertex count followed by the indices into the vertices array. If *normalCount* is not zero, this parameter should contain a vertex count followed by pairs of indices, with the first index of each pair indexing into the array of vertices, and the second indexing into the array of normals. The list of indices must terminate with a zero.

lplpD3DRMFaceArray
Address of a pointer to an *IDirect3DRMFaceArray* interface that will be filled with a pointer to the newly created faces.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::AddFrame

Adds the contents of a frame to a Direct3DRMMeshBuilder2 object.

HRESULT AddFrame(
LPDIRECT3DRMFRAME *lpD3DRMFrame*
);

Parameters

lpD3DRMFrame
Address of the frame whose contents are being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The source frame is not modified or referenced by this operation.

IDirect3DRMMeshBuilder2::AddMesh

Adds a mesh to a Direct3DRMMeshBuilder2 object.

HRESULT AddMesh(
LPDIRECT3DRMMESH *lpD3DRMMesh*
);

Parameters

lpD3DRMMesh
Address of the mesh being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::AddMeshBuilder

Adds the contents of a Direct3DRMMeshBuilder object to a Direct3DRMMeshBuilder2 object.

HRESULT AddMeshBuilder(
LPDIRECT3DRMMESHBUILDER *lpD3DRMMeshBuild*
);

Parameters

lpD3DRMMeshBuild
Address of the Direct3DRMMeshBuilder object whose contents are being added.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The source Direct3DRMMeshBuilder object is not modified or referenced by this operation.

IDirect3DRMMeshBuilder2::AddNormal

Adds a normal to a Direct3DRMMeshBuilder2 object.

int AddNormal(
D3DVALUE *x*,
D3DVALUE *y*,

D3DVALUE *z*
);

Parameters *x, y,* and *z*
The x, y, and z components of the direction of the new normal.

Return Values Returns the index of the normal.

IDirect3DRMMeshBuilder2::AddVertex

Adds a vertex to a Direct3DRMMeshBuilder2 object.

int AddVertex(
D3DVALUE *x***,**
D3DVALUE *y***,**
D3DVALUE *z*
);

Parameters *x, y,* and *z*
The x, y, and z components of the position of the new vertex.

Return Values Returns the index of the vertex.

IDirect3DRMMeshBuilder2::CreateFace

Creates a new face with no vertices and adds it to a Direct3DRMMeshBuilder2 object.

HRESULT CreateFace(
LPDIRECT3DRMFACE* *lplpD3DRMFace*
);

Parameters *lplpD3DRMFace*
Address of a pointer to an *IDirect3DRMFace* interface that will be filled with a pointer to the face that was created.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::CreateMesh

Creates a new mesh from a Direct3DRMMeshBuilder2 object.

HRESULT CreateMesh(
LPDIRECT3DRMMESH* *lplpD3DRMMesh*
);

Parameters

lplpD3DRMMesh
Address that will be filled with a pointer to an *IDirect3DRMMesh* interface.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::GenerateNormals2

Processes the Direct3DRMMeshBuilder2 object and generates normals for each vertex in a mesh by averaging the face normals for each face that shares the vertex. New normals are generated if the faces sharing a vertex have an angle between them greater than the crease angle.

HRESULT GenerateNormals2(
D3DVALUE *dvCreaseAngle*,
DWORD *dwFlags*
);

Parameters

dvCreaseAngle
The least angle in radians that faces can have between them and have a new normal generated.

dwFlags
One of the following values.

D3DRMGENERATENORMALS_PRECOMPACT (*dwFlags* = 1) – Specifies that the algorithm should attempt to compact mesh vertices before it generates normals. See comments below.

D3DRMGENERATENORMALS_USECREASEANGLE (*dwFlags* = 2) – Specifies that the *dvCreaseAngle* parameter should be used. Otherwise, the crease angle is ignored.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks If you specify the D3DRMGENERATENORMALS_PRECOMPACT flag, the precompact pass searches all the vertices in the mesh and merges any that are the same. This is a good way of compacting a mesh that has been loaded. Some meshes have multiple vertices if they have multiple normals at that vertex. This is not necessary in Direct3D Retained Mode. Specifying this flag is the way to get rid of the multiple vertices.

After compacting, the normals are generated. The edges between faces are examined, and if the angle the faces make at the edge is less than the crease angle, the face normals are averaged to generate the vertex normal. If the angle is greater than the crease angle, a new normal is generated. Note that a new vertex is not generated.

IDirect3DRMMeshBuilder2::GetBox

Retrieves the bounding box containing a Direct3DRMMeshBuilder2 object. The bounding box gives the minimum and maximum model coordinates in each dimension.

HRESULT GetBox(
D3DRMBOX **lpD3DRMBox*
);

Parameters *lpD3DRMBox*
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::GetColorSource

Retrieves the color source of a Direct3DRMMeshBuilder2 object. The color source can be either a face or a vertex.

D3DRMCOLORSOURCE GetColorSource( );

Return Values Returns a member of the **D3DRMCOLORSOURCE** enumerated type.

See Also **IDirect3DRMMeshBuilder2::SetColorSource**

IDirect3DRMMeshBuilder2::GetFace

Retrieves a single face of a Direct3DRMMeshBuilder2 object.

HRESULT GetFace(
DWORD *dwIndex*,
LPDIRECT3DRMFACE* *lplpD3DRMFace*
);

Parameters

dwIndex
The index of the mesh face to be retrieved. The face must already be part of a Direct3DRMMeshBuilder2 object.

lplpD3DRMFace
Address of a pointer to an *IDirect3DRMFace* interface that is filled with an address of the face.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::GetFaceCount

Retrieves the number of faces in a Direct3DRMMeshBuilder2 object.

int GetFaceCount( );

Return Values Returns the number of faces.

IDirect3DRMMeshBuilder2::GetFaces

Retrieves the faces of a Direct3DRMMeshBuilder2 object.

HRESULT GetFaces(
LPDIRECT3DRMFACEARRAY* *lplpD3DRMFaceArray*

);

Parameters

lplpD3DRMFaceArray
Address of a pointer to an *IDirect3DRMFaceArray* interface that is filled with an address of the faces.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::GetPerspective

Determines whether perspective correction is on for a Direct3DRMMeshBuilder2 object.

BOOL GetPerspective( );

Return Values

Returns TRUE if perspective correction is on, or FALSE otherwise.

IDirect3DRMMeshBuilder2::GetQuality

Retrieves the rendering quality of a Direct3DRMMeshBuilder2 object.

D3DRMRENDERQUALITY GetQuality( );

Return Values

Returns a member of the **D3DRMRENDERQUALITY** enumerated type that specifies the rendering quality of the mesh.

See Also

IDirect3DRMMeshBuilder2::SetQuality

IDirect3DRMMeshBuilder2::GetTextureCoordinates

Retrieves the texture coordinates of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT GetTextureCoordinates(
DWORD *index*,
D3DVALUE **lpU*,
D3DVALUE **lpV*
);

Parameters

index
Index of the vertex.

lpU and *lpV*
Addresses of variables that will be filled with the texture coordinates of the vertex when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMeshBuilder2::SetTextureCoordinates

IDirect3DRMMeshBuilder2::GetVertexColor

Retrieves the color of a specified vertex in a Direct3DRMMeshBuilder2 object.

D3DCOLOR GetVertexColor(
DWORD *index*
);

Parameters

index
Index of the vertex.

Return Values

Returns the color.

See Also

IDirect3DRMMeshBuilder2::SetVertexColor

IDirect3DRMMeshBuilder2::GetVertexCount

Retrieves the number of vertices in a Direct3DRMMeshBuilder2 object.

int GetVertexCount( );

Return Values

Returns the number of vertices.

IDirect3DRMMeshBuilder2::GetVertices

Retrieves the vertices, normals, and face data for a Direct3DRMMeshBuilder2 object.

HRESULT GetVertices(
DWORD **vcount*,
D3DVECTOR **vertices*,
DWORD **ncount*,
D3DVECTOR **normals*,
DWORD **face_data_size*,
DWORD **face_data*
);

Parameters

vcount
Address of a variable that will contain the number of vertices.

vertices
Address of an array of **D3DVECTOR** structures that will contain the vertices for the Direct3DRMMeshBuilder2 object. If this parameter is NULL, the method returns the number of vertices in the *vcount* parameter.

ncount
Address of a variable that will contain the number of normals.

normals
Address of an array of **D3DVECTOR** structures that will contain the normals for the Direct3DRMMeshBuilder2 object. If this parameter is NULL, the method returns the number of normals in the *ncount* parameter.

face_data_size
Address of a variable that specifies the size of the buffer pointed to by the *face_data* parameter. The size is given in units of **DWORD** values. This parameter cannot be NULL.

face_data
Address of the face data for the Direct3DRMMeshBuilder2 object. This data is in the same format as specified in the **IDirect3DRMMeshBuilder2::AddFaces** method except that it is null-terminated. If this parameter is NULL, the method returns the required size of the face-data buffer in the *face_data_size* parameter.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::Load

Loads a Direct3DRMMeshBuilder2 object.

HRESULT Load(
LPVOID *lpvObjSource*,
LPVOID *lpvObjID*,
D3DRMLOADOPTIONS *d3drmLOFlags*,
D3DRMLOADTEXTURECALLBACK *d3drmLoadTextureProc*,
LPVOID *lpvArg*
);

Parameters

lpvObjSource
Source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *d3drmLOFlags* parameter.

lpvObjID
Object name or position to be loaded. The use of this parameter depends on the identifier flags specified in the *d3drmLOFlags* parameter. If the D3DRMLOAD_BYPOSITION flag is specified, this parameter is a pointer to a **DWORD** value that gives the object's order in the file. This parameter can be NULL.

d3drmLOFlags
Value of the **D3DRMLOADOPTIONS** type describing the load options.

d3drmLoadTextureProc
A **D3DRMLOADTEXTURECALLBACK** callback function called to load any textures used by an object that require special formatting. This parameter can be NULL.

lpvArg
Address of application-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

By default, this method loads the first mesh from the source specified in the *lpvObjSource* parameter.

IDirect3DRMMeshBuilder2::ReserveSpace

Reserves space within a Direct3DRMMeshBuilder2 object for the specified number of vertices, normals, and faces. This allows the system to use memory more efficiently.

HRESULT ReserveSpace(
DWORD *vertexCount*,
DWORD *normalCount*,
DWORD *faceCount*
);

Parameters

vertexCount, normalCount, and *faceCount*
Number of vertices, normals, and faces to allocate space for.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::Save

Saves a Direct3DRMMeshBuilder2 object.

HRESULT Save(
const char * *lpFilename*,
D3DRMXOFFORMAT *d3drmXOFFormat*,
D3DRMSAVEOPTIONS *d3drmSOContents*
);

Parameters

lpFilename
Address specifying the name of the created file. This file must have a .X file name extension.

d3drmXOFFormat
The D3DRMXOF_TEXT value from the **D3DRMXOFFORMAT** enumerated type.

d3drmSOContents
Value of the **D3DRMSAVEOPTIONS** type describing the save options.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::Scale

Scales a Direct3DRMMeshBuilder2 object by the given scaling factors, parallel to the x-, y-, and z-axes in model coordinates.

HRESULT Scale(
D3DVALUE *sx*,
D3DVALUE *sy*,
D3DVALUE *sz*
);

Parameters

sx, sy, and *sz*
Scaling factors that are applied along the x-, y-, and z-axes.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetColor

Sets all the faces of a Direct3DRMMeshBuilder2 object to a given color.

HRESULT SetColor(
D3DCOLOR *color*
);

Parameters

color
Color of the faces.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetColorRGB

Sets all the faces of a Direct3DRMMeshBuilder2 object to a given color.

HRESULT SetColorRGB(
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*

);

Parameters *red, green,* and *blue*
Red, green, and blue components of the color.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetColorSource

Sets the color source of a Direct3DRMMeshBuilder2 object.

HRESULT SetColorSource(
D3DRMCOLORSOURCE *source*
);

Parameters *source*
Member of the **D3DRMCOLORSOURCE** enumerated type that specifies the new color source to use.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMMeshBuilder2::GetColorSource**

IDirect3DRMMeshBuilder2::SetMaterial

Sets the material of all the faces of a Direct3DRMMeshBuilder2 object.

HRESULT SetMaterial(
LPDIRECT3DRMMATERIAL *lpIDirect3DRMmaterial*
);

Parameters *lpIDirect3DRMmaterial*
Address of *IDirect3DRMMaterial* interface for the Direct3DRMMeshBuilder2 object.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetNormal

Sets the normal vector of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT SetNormal(
DWORD *index*,
D3DVALUE *x*,
D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

index
Index of the normal to be set.

x, y, and *z*
The x, y, and z components of the vector to assign to the specified normal.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetPerspective

Enables or disables perspective-correct texture-mapping for a Direct3DRMMeshBuilder2 object.

HRESULT SetPerspective(
BOOL *perspective*
);

Parameters

perspective
Specify TRUE if the mesh should be texture-mapped with perspective correction, or FALSE otherwise.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetQuality

Sets the rendering quality of a Direct3DRMMeshBuilder2 object.

HRESULT SetQuality(
D3DRMRENDERQUALITY *quality*
);

Parameters

quality
Member of the **D3DRMRENDERQUALITY** enumerated type that specifies the new rendering quality to use.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An object's quality has three components: shade mode (flat or gouraud, phong is not yet implemented and will default to gouraud shading), lighting type (on or off), and fill mode (point, wireframe or solid).

You can set the quality of a device with **IDirect3DRMDevice::SetQuality**. By default it is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMMeshBuilder2 object with the **SetQuality** method. By default, a Direct3DRMMeshBuilder2 object's quality is D3DRMRENDER_GOURAUD (gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

See Also

IDirect3DRMMeshBuilder2::GetQuality

IDirect3DRMMeshBuilder2::SetTexture

Sets the texture of all the faces of a Direct3DRMMeshBuilder2 object.

HRESULT SetTexture(
LPDIRECT3DRMTEXTURE *lpD3DRMTexture*
);

Parameters

lpD3DRMTexture
Address of the required Direct3DRMTexture object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetTextureCoordinates

Sets the texture coordinates of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT SetTextureCoordinates(
DWORD *index*,
D3DVALUE *u*,
D3DVALUE *v*
);

Parameters

index
Index of the vertex to be set.

u and *v*
Texture coordinates to assign to the specified mesh vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMeshBuilder2::GetTextureCoordinates

IDirect3DRMMeshBuilder2::SetTextureTopology

Sets the texture topology of a Direct3DRMMeshBuilder2 object.

HRESULT SetTextureTopology(
BOOL *cylU*,
BOOL *cylV*
);

Parameters

cylU and *cylV*
Specify TRUE for either or both of these parameters if you want the texture to have a cylindrical topology in the u and v dimensions respectively; otherwise, specify FALSE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetVertex

Sets the position of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT SetVertex(
DWORD *index*,
D3DVALUE *x*,
D3DVALUE *y*,
D3DVALUE *z*
);

Parameters

index
Index of the vertex to be set.

x, y, and *z*
The x, y, and z components of the position to assign to the specified vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::SetVertexColor

Sets the color of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT SetVertexColor(
DWORD *index*,
D3DCOLOR *color*
);

Parameters

index
Index of the vertex to be set.

color
Color to assign to the specified vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMMeshBuilder2::GetVertexColor

IDirect3DRMMeshBuilder2::SetVertexColorRGB

Sets the color of a specified vertex in a Direct3DRMMeshBuilder2 object.

HRESULT SetVertexColorRGB(
DWORD *index*,
D3DVALUE *red*,
D3DVALUE *green*,
D3DVALUE *blue*
);

Parameters

index
Index of the vertex to be set.

red, green, and *blue*
Red, green, and blue components of the color to assign to the specified vertex.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMMeshBuilder2::Translate

Adds the specified offsets to the vertex positions of a Direct3DRMMeshBuilder2 object.

HRESULT Translate(
D3DVALUE *tx*,
D3DVALUE *ty*,
D3DVALUE *tz*
);

Parameters

tx, ty, and *tz*
Offsets that are added to the x-, y-, and z-coordinates respectively of each vertex position.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObject

Applications use the methods of the **IDirect3DRMObject** interface to work with the object superclass of Direct3DRM objects. This section is a reference to the methods of this interface. For a conceptual overview, see *Direct3DRMObject*.

The methods of the **IDirect3DRMObject** interface can be organized into the following groups:

Application-specific data	**GetAppData** **SetAppData**
Cloning	**Clone**
Naming	**GetClassName** **GetName** **SetName**
Notifications	**AddDestroyCallback** **DeleteDestroyCallback**

The **IDirect3DRMObject** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMObject object is obtained through the appropriate call to the **QueryInterface** method from any Direct3DRM object. All Direct3DRM objects inherit the **IDirect3DRMObject** interface methods.

IDirect3DRMObject::AddDestroyCallback

Registers a function to be called when an object is destroyed.

```
HRESULT AddDestroyCallback(
    D3DRMOBJECTCALLBACK lpCallback,
    LPVOID lpArg
    );
```

Parameters

lpCallback
User-defined callback function that will be called when the object is destroyed.

lpArg
Address of application-defined data passed to the callback function. Because this function is called after the object has been destroyed, you should not call this function with the object as an argument.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObject::Clone

Creates a copy of an object.

```
HRESULT Clone(
    LPUNKNOWN pUnkOuter,
    REFIID riid,
    LPVOID *ppvObj
    );
```

Parameters

pUnkOuter
Allows COM aggregation features.

riid
Identifier of the object being copied.

ppvObj
Address that will contain the copy of the object when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObject::DeleteDestroyCallback

Removes a function previously registered with the **IDirect3DRMObject::AddDestroyCallback** method.

HRESULT DeleteDestroyCallback(
D3DRMOBJECTCALLBACK *d3drmObjProc*,
LPVOID *lpArg*
);

Parameters

d3drmObjProc
User-defined **D3DRMOBJECTCALLBACK** callback function that will be called when the object is destroyed.

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

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObject::GetAppData

Retrieves the 32 bits of application-specific data in the object. The default value is 0.

DWORD GetAppData( );

Return Values Returns the data value defined by the application.

See Also **IDirect3DRMObject::SetAppData**

IDirect3DRMObject::GetClassName

Retrieves the name of the object's class.

HRESULT GetClassName(
LPDWORD *lpdwSize*,
LPSTR *lpName*
);

Parameters

lpdwSize
Address of a variable containing the size, in bytes, of the buffer pointed to by the *lpName* parameter.

lpName
Address of a variable that will contain a null-terminated string identifying the class name when the method returns. If this parameter is NULL, the *lpdwSize* parameter will contain the required size for the string when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMObject::GetName

Retrieves the object's name.

HRESULT GetName(
LPDWORD *lpdwSize*,
LPSTR *lpName*
);

Parameters

lpdwSize
Address of a variable containing the size, in bytes, of the buffer pointed to by the *lpName* parameter.

lpName
Address of a variable that will contain a null-terminated string identifying the object's name when the method returns. If this parameter is NULL, the *lpdwSize* parameter will contain the required size for the string when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMObject::SetName

IDirect3DRMObject::SetAppData

Sets the 32 bits of application-specific data in the object.

HRESULT SetAppData(
DWORD *ulData*
);

Parameters

ulData
User-defined data to be stored with the object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMObject::GetAppData

IDirect3DRMObject::SetName

Sets the object's name.

HRESULT SetName(
const char * *lpName*
);

Parameters

lpName
User-defined data to be the name for the object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMObject::GetName

IDirect3DRMProgressiveMesh

The Progressive Mesh is a form of mesh that allows for progressive refinement. Conceptually, it consists of a base mesh representation and a number of "vertex split" records. A mesh can be stored as a much coarser mesh together with records of how to incrementally refine the mesh. This allows for a generalized level of detail to be set on the mesh as well as progressive download of the mesh from a remote source.

The progressive mesh is a visual in Direct3D Retained Mode and may be used in the standard ways that visuals are used. That is, a progressive mesh can be added to frame hierarchies, multiply instanced and picked.

The **Direct3DRMProgressiveMesh** object is created by calling the **IDirect3DRM2::CreateProgressiveMesh** method. After creation, the object can be added to a hierarchy, but will not render until at least the base mesh is available.

For a conceptual overview, see *IDirect3DRMProgressiveMesh*.

In addition to the standard *IUnknown* and **IDirect3DRMObject** methods, this API contains the following members:

Creating and Copying Meshes	**Clone**
	CreateMesh
	Duplicate
	GetBox
Loading	**Abort**
	GetLoadStatus
	Load
Setting Quality	**SetQuality**
	GetQuality
Managing Details	**GetDetail**
	GetFaceDetail
	GetFaceDetailRange
	GetVertexDetail
	GetVertexDetailRange
	SetDetail
	SetFaceDetail
	SetMinRenderDetail
	SetVertexDetail
Registering Events	**RegisterEvents**

The **IDirect3DRMProgressiveMesh** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMProgressiveMesh** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

IDirect3DRMProgressiveMesh::Abort

Terminates the currently active download.

HRESULT Abort(
DWORD *dwFlags*
)

Parameters

dwFlags
Must be set to zero.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

Remarks

If the base mesh has been downloaded before this method is called, the effect is as if the progressive mesh has loaded, the vertex splits are in a valid state, and the progressive mesh is renderable. The other progressive mesh methods will work. If the base mesh has not been downloaded before the **IDirect3DRMProgressiveMesh::Abort** call and you have added the progressive mesh to a scene, the **Render** will succeed but the progressive mesh will not be rendered. Also, if the base mesh has not been downloaded, when you try to use the progressive mesh (to create a mesh or clone, for example) the call will return D3DRMERR_NOTENOUGHDATA.

If there are any outstanding events, they are signaled.

IDirect3DRMProgressiveMesh::Clone

Creates a copy of the currently loaded **Direct3DRMProgressiveMesh** object.

HRESULT Clone(
LPDIRECT3DRMPROGRESSIVEMESH* *lplpD3DRMPMesh*
)

Parameters

lplpD3DRMPMesh
Address of a **Direct3DRMProgressiveMesh** pointer that will be filled in with a pointer to the newly generated **Direct3DRMProgressiveMesh** object.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_NOTENOUGHDATA

Remarks

The progressive mesh being cloned must have at least its base mesh loaded. If you call this method on a progressive mesh that is currently being asynchronously loaded, the cloned mesh only has as much detail as the loading progressive mesh had at the time it was cloned. Any vertex splits loaded after the duplication are not available to the cloned mesh.

This method does not try to share any of the progressive mesh's internal data, whereas the **IDirect3DRMProgressiveMesh::Duplicate** does.

See Also

IDirect3DRMProgressiveMesh::Duplicate

IDirect3DRMProgressiveMesh::CreateMesh

Builds a mesh from the current level of detail.

HRESULT CreateMesh(
LPDIRECT3DRMMESH* *lplpD3DRMMesh*
)

Parameters

lplpD3DRMMesh
Address of a **Direct3DRMMesh** pointer that will be filled in with a pointer to the newly generated **Direct3DRMMesh** object.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_NOTENOUGHDATA

Remarks

If the application has requested a level of detail that is not yet available, or the base mesh is not yet available, this method returns the error **D3DRMERR_NOTENOUGHDATA**.

IDirect3DRMProgressiveMesh::Duplicate

Creates a copy of the **Direct3DRMProgressiveMesh** object. The copy shares all geometry and face data with the original, but has a detail level that can be set independently of the original. This enables the same mesh data to be used in different parts of the hierarchy but with different levels of detail. In essence, you almost have two instances of a progressive mesh within the frame hierarchy.

HRESULT Duplicate(
LPDIRECT3DRMPROGRESSIVEMESH* *lplpD3DRMPMesh*
)

Parameters

lplpD3DRMPMesh
Address of a **Direct3DRMProgressiveMesh** pointer that will be filled with a pointer to the newly generated **Direct3DRMProgressiveMesh** object.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_NOTENOUGHDATA

Remarks

The progressive mesh being duplicated must have at least its base mesh loaded. If you call this method on a progressive mesh that is currently being asynchronously loaded, the duplicated mesh only has as much detail as the loading progressive

mesh had at the time it was duplicated. Any vertex splits loaded after the duplication are not available to the duplicated mesh.

A progressive mesh has a set of data representing the base mesh, and a set of data representing the vertex splits. The base mesh data and data describing the current state of the progressive mesh and the current level of detail isn't shared between meshes that are duplicated, but the vertex splits are. This method creates a new instance of the Progressive Mesh which shares all geometry and face data with the original, but has a detail level that can be set independently of the original.

IDirect3DRMProgressiveMesh::GetBox

Retrieves the bounding box containing a Direct3DRMProgressiveMesh object. The bounding box gives the minimum and maximum coordinates relative to a child frame, in each dimension.

HRESULT GetBox(
D3DRMBOX * *lpD3DRMBox*
);

Parameters

lpD3DRMBox
Address of a **D3DRMBOX** structure that will be filled with the bounding box coordinates.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMProgressiveMesh::GetDetail

Returns the current detail level of the progressive mesh normalized between 0.0 and 1.0.

HRESULT GetDetail(
LPD3DVALUE *lpdvVal*
)

Parameters

lpdvVal
Address of a **D3DVALUE** that will be filled with the current detail level of the progressive mesh.

Return Values Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_PENDING

Remarks If the base mesh has not yet downloaded then this method will return **D3DRMERR_PENDING**. If a requested level of detail has been set, the return value will increase on each subsequent call until the requested level has been met. If no level has been requested, then the detail will increase until all vertex splits have been downloaded.

The normalized value 0.0 represents the minimum number of vertices (the number of vertices in the base mesh), and the normalized value 1.0 represents the maximum number of vertices.

IDirect3DRMProgressiveMesh::GetFaceDetail

Retrieves the number of faces in the progressive mesh.

HRESULT GetFaceDetail(
LPDWORD *lpdwCount*
)

Parameters *lpdwCount*
Address of a **DWORD** that will be filled in with the number of faces in the progressive mesh.

Return Values Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_PENDING
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS

Remarks If the number of faces is not available, this method returns **D3DRMERR_PENDING**.

IDirect3DRMProgressiveMesh::GetFaceDetailRange

Retrieves the minimum and maximum face count available in the progressive mesh.

HRESULT GetFaceDetailRange(
LPDWORD *lpdwMinFaces*,
LPDWORD *lpdwMaxFaces*
)

Parameters

lpdwMinFaces
Address of a **DWORD** that will be filled in with the minimum number of faces.

lpdwMaxFaces
Address of a **DWORD** that will be filled in with the maximum number of faces.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_PENDING

Remarks

If the face count is not available, this method returns **D3DRMERR_PENDING**.

IDirect3DRMProgressiveMesh::GetLoadStatus

Allows the application to inquire about the current status of the load.

HRESULT GetLoadStatus(
LPD3DRMPMESHLOADSTATUS *lpStatus*
)

Parameters *lpStatus*

Address of a D3DRMPMESHLOADSTATUS structure defined as follows:

```
typedef struct _D3DRMPMESHLOADSTATUS
{
    DWORD dwSize;           // Size of this structure
    DWORD dwPMeshSize;      // Total size (bytes)
    DWORD dwBaseMeshSize;       // Size of base mesh (bytes)
    DWORD dwBytesLoaded;    // Total number of bytes loaded
    DWORD dwVerticesLoaded;     // Number of vertices loaded
    DWORD dwFacesLoaded;    // Number of faces loaded
    DWORD dwFlags;
}
D3DRMPMESHLOADSTATUS;
typedef D3DRMPMESHLOADSTATUS *LPD3DRMPMESHLOADSTATUS;
```

The *dwFlags* member can take the following values:

D3DRMPMESHSTATUS_VALID – The progressive mesh object contains valid data.

D3DRMPMESHSTATUS_INTERRUPTED – The download was interrupted either because the application called **Abort** or because the connection was lost.

D3DRMPMESHSTATUS_BASEMESH – The base mesh has been downloaded.

D3DRMPMESHSTATUS_COMPLETE – All data has been downloaded.

D3DRMPMESHSTATUS_RENDERABLE – It is now possible to render the mesh.

Return Values Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST

D3DRMERR_INVALIDDATA

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

Remarks If the mesh is renderable (the base mesh has been downloaded and the data is not corrupt) then the *dwFlags* member contains D3DRMPMESHSTATUS_RENDERABLE. If the download was interrupted, the *dwFlags* member will contain D3DRMPMESHSTATUS_INTERRUPTED.

IDirect3DRMProgressiveMesh::GetQuality

Retrieves a member of the **D3DRMRENDERQUALITY** enumerated type that specifies the rendering quality of the progressive mesh.

HRESULT GetQuality(
LPD3DRMRENDERQUALITY *lpQuality*
);

Parameters

lpQuality
Pointer to the **D3DRMRENDERQUALITY** member if successful that specifies rendering quality.

Return Values

Returns D3DRM_OK if successful, or one of the following errors:

D3DRMERR_BADOBJECT - the progressive mesh is invalid.

D3DRMERR_BADVALUE - the pointer to the **D3DRMRENDERQUALITY** member is invalid.

See Also

IDirect3DRMProgressiveMesh::SetQuality

IDirect3DRMProgressiveMesh::GetVertexDetail

Retrieves the number of vertices in the progressive mesh.

HRESULT GetVertexDetail(
LPDWORD *lpdwCount*
)

Parameters

lpdwCount
Address of a **DWORD** that will be filled in with the number of vertices in the progressive mesh.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST

D3DRMERR_INVALIDDATA

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

D3DRMERR_PENDING

Remarks If the number of vertices is not available, this method returns **D3DRMERR_PENDING**.

IDirect3DRMProgressiveMesh::GetVertexDetailRange

Retrieves the minimum and maximum vertex count available in the progressive mesh.

HRESULT GetVertexDetailRange(
LPDWORD *lpdwMinVertices***,**
LPDWORD *lpdwMaxVertices*
)

Parameters

lpdwMinVertices
Address of a **DWORD** that will be filled in with the minimum number of vertices.

lpdwMaxVertices
Address of a **DWORD** that will be filled in with the maximum number of vertices.

Return Values Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST

D3DRMERR_INVALIDDATA

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

D3DRMERR_PENDING

Remarks If the vertex count information is not available, this method returns **D3DRMERR_PENDING**.

IDirect3DRMProgressiveMesh::Load

Loads the progressive mesh object from memory, a file, a resource, or a URL. Loading can be done synchronously or asynchronously.

HRESULT Load(

LPVOID *lpSource*,
LPVOID *lpObjID*,
D3DRMLOADOPTIONS *dloLoadflags*,
D3DRMLOADTEXTURECALLBACK *lpCallback*,
LPVOID *lpArg*
)

Parameters

lpSource

Address of the source for the object to be loaded. This source can be a file, resource, memory block, or stream, depending on the source flags specified in the *dloLoadflags* parameter.

lpObjID

Address of the ID of the DirectX file record that is a Progressive Mesh. This can either be a string or a UUID (determined by *dloLoadflags*). If *lpObjID* is NULL, then *dloLoadflags* must be **D3DRMLOAD_FIRST**.

dloLoadflags

Value of the **D3DRMLOADOPTIONS** type describing how the load is to be performed. One flag from each of the following two groups must be included:

The following values determine which object in the DirectX file is loaded:	
D3DRMLOAD_BYNAME	The *lpObjID* parameter is interpreted as a string.
D3DRMLOAD_BYGUID	The *lpObjID* parameter is interpreted as a UUID.
D3DRMLOAD_FIRST	The first progressive mesh found is loaded.
The following flags determine the source of the DirectX file:	
D3DRMLOAD_FROMFILE	The *lpSource* parameter is interpreted as a string representing a local file name.
D3DRMLOAD_FROMRESOURCE	The *lpSource* parameter is interpreted as a pointer to a **D3DRMLOADRESOURCE** structure.
D3DRMLOAD_FROMMEMORY	The *lpSource* parameter is interpreted as a pointer to a **D3DRMLOADMEMORY** structure.
D3DRMLOAD_FROMURL	The *lpSource* parameter is interpreted as a URL.

In addition, you can specify whether the download is synchronous or asynchronous. By default, loading is synchronous and the **Load** call will not return until all data has been loaded or an error occurs. To specify asynchronous loading, include the following flag:

D3DRMLOAD_ASYNCHRONOUS – The **Load** call will return immediately. It is up to the application to use events with **IDirect3DRMProgressiveMesh::RegisterEvents** and **IDirect3DRMProgressiveMesh::GetLoadStatus** to find out how the load is progressing.

lpCallback
Address of a **D3DRMLOADTEXTURECALLBACK** callback function that will be called to load any texture encountered. The callback will be called with the texture name as encountered by the loader and the user-defined *lpArg* parameter. A new thread is not spawned for the callback. If you want the application to download a texture progressively, it must spawn a thread and return with a LPDIRECT3DRMTEXTURE as normal.

lpArg
Address of user-defined data passed to the **D3DRMLOADTEXTURECALLBACK** callback function.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_BADPMDATA
D3DRMERR_BADFILE
D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS

Remarks

The progressive mesh is not useful unless it has been initialized. If asynchronous download is specified, the API returns immediately and a separate thread is spawned to perform the download.

This method, **IDirect3DRMProgressiveMesh::Clone**, and **IDirect3DRMProgressiveMesh::Duplicate** all initialize a progressive mesh. You can only initialize an object once. Therefore, you cannot clone or duplicate a progressive mesh and then try to load into the cloned or duplicated mesh, nor can you load into a previously loaded mesh.

IDirect3DRMProgressiveMesh::RegisterEvents

Allows the application to register events with the progressive mesh object that will be signaled when the appropriate conditions are met.

HRESULT RegisterEvents(
HANDLE *hEvent*,
DWORD *dwFlags*,
DWORD *dwReserved*
)

Parameters

hEvent
Event to be signaled when the required condition is met.

dwFlags
Can be one of the following flags:

D3DRMPMESHEVENT_BASEMESH	The event is signaled when the base mesh has been downloaded.
D3DRMPMESHEVENT_COMPLETE	The event is signaled when all data has been downloaded.

dwReserved
Must be zero.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS

Remarks

This method can be used to monitor the progress of loads. Events will also be signaled if an error occurs, so your application should always call the **IDirect3DRMProgressiveMesh::GetLoadStatus** method after being signaled.

IDirect3DRMProgressiveMesh::SetDetail

Sets a requested level of detail normalized between 0.0 and 1.0.

HRESULT SetDetail(
D3DVALUE *dvVal*
)

Parameters

dvVal
The requested level of detail.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_BADPMDATA

D3DRMERR_CONNECTIONLOST

D3DRMERR_INVALIDDATA

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

D3DRMERR_PENDING

Remarks

If not enough detail has been downloaded yet (but will be available), the request is acknowledged and D3DRMERR_PENDING is returned. This error is informational and simply indicates that the requested level will be set as soon as enough detail is available.

The normalized value 0.0 represents the minimum number of vertices (the number of vertices in the base mesh), and the normalized value 1.0 represents the maximum number of vertices.

IDirect3DRMProgressiveMesh::SetFaceDetail

Sets the requested level of face detail.

HRESULT SetFaceDetail(
DWORD *dwCount*
)

Parameters

dwCount
The number of faces requested.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_BADPMDATA

D3DRMERR_CONNECTIONLOST

D3DRMERR_INVALIDDATA

D3DRMERR_INVALIDOBJECT

D3DRMERR_INVALIDPARAMS

D3DRMERR_PENDING

D3DRMERR_REQUESTTOOLARGE

Remarks

Sometimes it is not possible to set the progressive mesh to the number of faces requested, though it will always be within 1 of the requested value. This is because a vertex split can add 1 or 2 faces. For example, if you call **SetFaceDetail**(20), the progressive mesh may only be able to set the face detail to

19 or 21. You can always get the actual number of faces in the progressive mesh by calling the **IDirect3DRMProgressiveMesh::GetFaceDetail** method.

If not enough detail has been downloaded yet (but will be available), the request is be acknowledged and D3DRMERR_PENDING is returned. This error is informational and simply indicates that the requested level will be set as soon as enough detail is available. If the detail requested is greater that the detail available in the progressive mesh then D3DRMERR_REQUESTTOOLARGE is returned.

See Also **IDirect3DRMProgressiveMesh::GetFaceDetail**

IDirect3DRMProgressiveMesh::SetMinRenderDetail

Sets the minimum level of detail that will be rendered during a load from 0.0 (minimum detail) to 1.0 (maximum detail). Normally, the progressive mesh will be rendered once the base mesh is available (and the mesh is in the scene graph).

HRESULT SetMinRenderDetail(
D3DVALUE *dvCount*
)

Parameters

dvCount
The requested minimum detail.

Return Values

Returns DD_OK if successful, or one of the following errors:

D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_PENDING
D3DRMERR_REQUESTTOOLARGE
D3DRMERR_REQUESTTOOSMALL

Remarks

This API sets the minimum number of faces/vertices that will be rendered during the load (larger than the base mesh).

Any subsequent **IDirect3DRMProgressiveMesh::SetDetail**, **IDirect3DRMProgressiveMesh::SetFaceDetail**, or **IDirect3DRMProgressiveMesh::SetVertexDetail** calls will override this.

IDirect3DRMProgressiveMesh::SetQuality

Sets the rendering quality of a Direct3DRMProgressiveMesh object.

HRESULT SetQuality(
D3DRMRENDERQUALITY *quality*
);

Parameters

quality
Member of the **D3DRMRENDERQUALITY** enumerated type that specifies the new rendering quality to use.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

An object's quality has three components: shade mode (flat or gouraud, phong is not yet implemented and will default to gouraud shading), lighting type (on or off), and fill mode (point, wireframe or solid).

You can set the quality of a device with **IDirect3DRMDevice::SetQuality**. By default it is D3DRMRENDER_FLAT (flat shading, lights on, and solid fill).

You can set the quality of a Direct3DRMProgressiveMesh object with the **SetQuality** method. By default, a Direct3DRMProgressiveMesh object's quality is D3DRMRENDER_GOURAUD (gouraud shading, lights on, and solid fill).

DirectX Retained Mode renders an object at the lowest quality setting based on the device and object's current setting for each individual component. For example, if the object's current quality setting is D3DRMRENDER_GOURAUD, and the device is D3DRMRENDER_FLAT then the object will be rendered with flat shading, solid fill and lights on.

If the object's current quality setting is D3DRMSHADE_GOURAUD| D3DRMLIGHT_OFF|D3DRMFILL_WIREFRAME and the device's quality setting is D3DRMSHADE_FLAT|D3DRMLIGHT_ON|D3DRMFILL_POINT, then the object will be rendered with flat shading, lights off and point fill mode.

These rules apply to Direct3DRMMeshBuilder objects, Direct3DRMMeshBuilder2 objects, and Direct3DRMProgressiveMesh objects. However, Direct3DRMMesh objects do not follow these rules. Mesh objects

ignore the device's quality settings and use the group quality setting (which defaults to D3DRMRENDER_GOURAUD).

See Also **IDirect3DRMProgressiveMesh::GetQuality**

IDirect3DRMProgressiveMesh::SetVertexDetail

Sets the requested level of vertex detail.

HRESULT SetVertexDetail(
DWORD *dwCount*
)

Parameters

dwCount
The number of vertices requested.

Return Values Returns DD_OK if successful, or one of the following errors:

D3DRMERR_BADPMDATA
D3DRMERR_CONNECTIONLOST
D3DRMERR_INVALIDDATA
D3DRMERR_INVALIDOBJECT
D3DRMERR_INVALIDPARAMS
D3DRMERR_PENDING
D3DRMERR_REQUESTTOOLARGE

Remarks If not enough detail has been downloaded yet (but will be available), the request will be acknowledged and D3DRMERR_PENDING is returned. This error is informational and simply indicates that the requested level will be set as soon as enough detail is available. If the detail requested is greater that the detail available in the progressive mesh then D3DRMERR_REQUESTTOOLARGE is returned.

See Also **IDirect3DRMProgressiveMesh::GetVertexDetail**

IDirect3DRMShadow

Applications use the **IDirect3DRMShadow** interface to initialize Direct3DRMShadow objects. Note that this initialization is not necessary if the

application calls the **IDirect3DRM::CreateShadow** method; it is required only if the application calls the **IDirect3DRM::CreateObject** method to create the shadow.

This section is a reference to the methods of the **IDirect3DRMShadow** interface. For a conceptual overview, see *IDirect3DRMShadow Interface*.

The **IDirect3DRMShadow** interface supports the **Init** method.

The **IDirect3DRMShadow** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMShadow** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMShadow object is obtained by calling the **IDirect3DRM::CreateShadow** method.

IDirect3DRMShadow::Init

Initializes a Direct3DRMShadow object.

HRESULT Init(
LPDIRECT3DRMVISUAL *lpD3DRMVisual*,
LPDIRECT3DRMLIGHT *lpD3DRMLight*,
D3DVALUE *px*,
D3DVALUE *py*,

D3DVALUE *pz*,
D3DVALUE *nx*,
D3DVALUE *ny*,
D3DVALUE *nz*
);

Parameters

lpD3DRMVisual
Address of the Direct3DRMVisual object casting the shadow.

lpD3DRMLight
Address of the Direct3DRMLight object that provides the light that defines the shadow.

px, *py*, and *pz*
Coordinates of a point on the plane on which the shadow is cast.

nx, *ny*, and *nz*
Coordinates of the normal vector of the plane on which the shadow is cast.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMTexture

Applications use the methods of the **IDirect3DRMTexture** interface to work with textures, which are rectangular arrays of pixels. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMTexture and IDirect3DRMTexture2 Interfaces*.

To avoid unnecessary delays when creating textures, hold onto textures you want to use again, instead of creating them each time they're needed. For optimal performance, use a texture surface format that is supported by the device you are using. This will avoid a costly format conversion when the texture is created and any time it changes.

See the remarks in **IDirect3DRM::LoadTexture** for an example showing how to keep a reference to textures loaded in a texture callback through **IDirect3DRM::LoadTexture**.

The methods of the **IDirect3DRMTexture** interface can be organized into the following groups:

Color	**GetColors**
	SetColors
Decals	**GetDecalOrigin**
	GetDecalScale

	GetDecalSize
	GetDecalTransparency
	GetDecalTransparentColor
	SetDecalOrigin
	SetDecalScale
	SetDecalSize
	SetDecalTransparency
	SetDecalTransparentColor
Images	**GetImage**
Initialization	**InitFromFile**
	InitFromResource
	InitFromSurface
Renderer notification	**Changed**
Shading	**GetShades**
	SetShades

The **IDirect3DRMTexture** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMTexture** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMTexture object is obtained by calling the **IDirect3DRM::CreateTexture** method.

IDirect3DRMTexture::Changed

Informs the renderer that the application has changed the pixels or the palette of a texture.

HRESULT Changed(
BOOL *bPixels*,
BOOL *bPalette*
);

Parameters

bPixels
If this parameter is TRUE, the pixels have changed.

bPalette
If this parameter is TRUE, the palette has changed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMTexture::GetColors

Retrieves the maximum number of colors used for rendering a texture.

DWORD GetColors( );

Return Values

Returns the number of colors.

Remarks

This method returns the number of colors that the texture has been quantized to, not the number of colors in the image from which the texture was created. Consequently, the number of colors that are returned usually matches the colors that were set by calling the **IDirect3DRM::SetDefaultTextureColors** method, unless you used the **IDirect3DRMTexture::SetColors** method explicitly to change the colors for the texture.

See Also

IDirect3DRMTexture::SetColors

IDirect3DRMTexture::GetDecalOrigin

Retrieves the current origin of the decal.

HRESULT GetDecalOrigin(
LONG * *lplX*,
LONG * *lplY*
);

Parameters

lplX and *lplY*
Addresses of variables that will be filled with the origin of the decal when the method returns.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMTexture::SetDecalOrigin

IDirect3DRMTexture::GetDecalScale

Retrieves the scaling property of the given decal.

DWORD GetDecalScale( );

Return Values

Returns the scaling property if successful, or -1 otherwise.

See Also

IDirect3DRMTexture::SetDecalScale

IDirect3DRMTexture::GetDecalSize

Retrieves the size of the decal.

HRESULT GetDecalSize(
D3DVALUE * *lprvWidth*,
D3DVALUE * *lprvHeight*
);

Parameters *lprvWidth* and *lprvHeight*
Addresses of variables that will be filled with the width and height of the decal when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture::SetDecalSize**

IDirect3DRMTexture::GetDecalTransparency

Retrieves the transparency property of the decal.

BOOL GetDecalTransparency( );

Return Values Returns TRUE if the decal has a transparent color, FALSE otherwise.

See Also **IDirect3DRMTexture::SetDecalTransparency**

IDirect3DRMTexture::GetDecalTransparentColor

Retrieves the transparent color of the decal.

D3DCOLOR GetDecalTransparentColor( );

Return Values Returns the value of the transparent color.

See Also **IDirect3DRMTexture::SetDecalTransparentColor**

IDirect3DRMTexture::GetImage

Returns an address of the image that the texture was created with. Returns a NULL pointer if the current texture was created with a DirectDraw surface.

D3DRMIMAGE * GetImage( );

Return Values Returns the address of the **D3DRMIMAGE** structure that the current texture was created with.

IDirect3DRMTexture::GetShades

Retrieves the number of shades used for each color in the texture when rendering.

DWORD GetShades( );

Return Values Returns the number of shades.

See Also **IDirect3DRMTexture::SetShades**

IDirect3DRMTexture::InitFromFile

Initializes a texture by using the information in a given file.

HRESULT InitFromFile(
const char **filename*
);

Parameters *filename*
Address of a string specifying the file from which initialization information is drawn.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks You must have created the texture to be initialized using the **IDirect3DRM::CreateObject** method.

See Also **IDirect3DRMTexture::InitFromResource**, **IDirect3DRMTexture::InitFromSurface**

IDirect3DRMTexture::InitFromResource

Initializes a Direct3DRMTexture object from a specified resource.

HRESULT InitFromResource(
HRSRC *rs*
);

Parameters

rs
Handle of the specified resource.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMTexture::InitFromFile, **IDirect3DRMTexture::InitFromSurface**

IDirect3DRMTexture::InitFromSurface

Initializes a texture by using the data from a given DirectDraw surface.

HRESULT InitFromSurface(
LPDIRECTDRAWSURFACE *lpDDS*
);

Parameters

lpDDS
Address of a DirectDraw surface from which initialization information is drawn.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMTexture::InitFromFile, **IDirect3DRMTexture::InitFromResource**

IDirect3DRMTexture::SetColors

Sets the maximum number of colors used for rendering a texture. This method is required only in the ramp color model.

HRESULT SetColors(
DWORD *ulColors*
);

Parameters *ulColors*
Number of colors. The default value is 8.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture::GetColors**

IDirect3DRMTexture::SetDecalOrigin

Sets the origin of the decal as an offset from the top left of the decal.

```
HRESULT SetDecalOrigin(
    LONG lX,
    LONG lY
    );
```

Parameters *lX* and *lY*
New origin, in decal coordinates, for the decal. The default origin is [0, 0].

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The decal's origin is mapped to its frame's position when rendering. For example, the origin of a decal of a cross would be set to the middle of the decal, and the origin of an arrow pointing down would be set to midway along the bottom edge.

This method is also used to add a decal origin key to a Direct3DRMTextureInterpolator object.

See Also **IDirect3DRMTexture::GetDecalOrigin**

IDirect3DRMTexture::SetDecalScale

Sets the scaling property for a decal.

```
HRESULT SetDecalScale(
    DWORD dwScale
    );
```

Parameters *dwScale*
If this parameter is TRUE, depth is taken into account when the decal is scaled. If it is FALSE, depth information is ignored. The default value is TRUE.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture::GetDecalScale**

IDirect3DRMTexture::SetDecalSize

Sets the size of the decal to be used if the decal is being scaled according to its depth in the scene.

HRESULT SetDecalSize(
D3DVALUE *rvWidth*,
D3DVALUE *rvHeight*
);

Parameters *rvWidth* and *rvHeight*
New width and height, in model coordinates, of the decal. The default size is [1, 1].

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a decal size key to a Direct3DRMTextureInterpolator object.

See Also **IDirect3DRMTexture::GetDecalSize**

IDirect3DRMTexture::SetDecalTransparency

Sets the transparency property of the decal.

HRESULT SetDecalTransparency(
BOOL *bTransp*
);

Parameters *bTransp*
If this parameter is TRUE, the decal has a transparent color. If it is FALSE, it has an opaque color. The default value is FALSE.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture::GetDecalTransparency**

IDirect3DRMTexture::SetDecalTransparentColor

Sets the transparent color for a decal.

HRESULT SetDecalTransparentColor(
D3DCOLOR *rcTransp*
);

Parameters *rcTransp*
New transparent color. The default transparent color is black.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a decal transparent color key to a Direct3DRMTextureInterpolator object.

See Also **IDirect3DRMTexture::GetDecalTransparentColor**

IDirect3DRMTexture::SetShades

Sets the maximum number of shades to use for each color for the texture when rendering. This method is required only in the ramp color model.

HRESULT SetShades(
DWORD *ulShades*
);

Parameters *ulShades*
New number of shades. This value must be a power of 2. The default value is 16.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture::GetShades**

IDirect3DRMTexture2

The **IDirect3DRMTexture2** interface is an extension of the *IDirect3DRMTexture* interface. **IDirect3DRMTexture2::InitFromResource2** allows resources to be loaded from DLLs and executables other than the currently executing file. In addition, **IDirect3DRMTexture2** has two new methods. The **IDirect3DRMTexture2::InitFromImage** method creates a texture from an image in memory. This method is equivalent to the **IDirect3DRM::CreateTexture** method. The **IDirect3DRMTexture2::GenerateMIPMap** method generates a mipmap from a source image.

To avoid unnecessary delays when creating textures, hold onto textures you want to use again, instead of creating them each time they're needed. For optimal performance, use a texture surface format that is supported by the device you are using. This will avoid a costly format conversion when the texture is created and any time it changes.

See the remarks in **IDirect3DRM2::LoadTexture** for an example showing how to keep a reference to textures loaded in a texture callback through **IDirect3DRM2::LoadTexture**.

For a conceptual overview, see *IDirect3DRMTexture and IDirect3DRMTexture2*.

The methods of the **IDirect3DRMTexture2** interface can be organized into the following groups:

Color	**GetColors**
	SetColors
Decals	**GetDecalOrigin**
	GetDecalScale
	GetDecalSize
	GetDecalTransparency
	GetDecalTransparentColor
	SetDecalOrigin
	SetDecalScale
	SetDecalSize
	SetDecalTransparency
	SetDecalTransparentColor
Images	**GetImage**

Initialization	**InitFromFile** **InitFromImage** **InitFromResource2** **InitFromSurface**
MIP map generation	**GenerateMIPMap**
Renderer notification	**Changed**
Shading	**GetShades** **SetShades**

The **IDirect3DRMTexture2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMTexture2** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

You can create an IDirect3DRMTexture2 object by calling **IDirect3DRM2::CreateTexture** or **IDirect3DRM2::CreateTextureFromSurface**.

IDirect3DRMTexture2::Changed

Informs the renderer that the application has changed the pixels or the palette of a Direct3DRMTexture2 object.

HRESULT Changed(
BOOL *bPixels*,
BOOL *bPalette*
);

Parameters

bPixels
If this parameter is TRUE, the pixels have changed.

bPalette
If this parameter is TRUE, the palette has changed.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMTexture2::GenerateMIPMap

Generates a mipmap from a single image source.

HRESULT GenerateMIPMap(
DWORD *dwFlags*
);

Parameters

dwFlags
Should be set to zero.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method can be called any time after a texture is created. It will generate a MIP map of the source image down to a resolution of 1×1 by using bi-linear filtering between levels. Once a MIP map has been generated, it will always be available and will be updated whenever **IDirect3DRMTexture::Changed** is called. When using MIP mapping, remember to change the texture quality to D3DRMTEXTURE_MIPNEAREST, D3DRMTEXTURE_MIPLINEAR, D3DRMTEXTURE_LINEARMIPNEAREST, or D3DRMTEXTURE_LINEARMIPLINEAR using **IDirect3DRMDevice::SetTextureQuality**. Extra MIP map levels will not be put

into video memory for hardware devices unless the texture quality includes a MIP mapping type and the hardware device supports MIP mapping.

See Also *Mipmaps*

IDirect3DRMTexture2::GetColors

Retrieves the maximum number of colors used for rendering a Direct3DRMTexture2 object.

DWORD GetColors( );

Return Values Returns the number of colors.

Remarks This method returns the number of colors that the Direct3DRMTexture2 object has been quantized to, not the number of colors in the image from which the Direct3DRMTexture2 object was created. Consequently, the number of colors that are returned usually matches the colors that were set by calling the **IDirect3DRM::SetDefaultTextureColors** method, unless you used the **IDirect3DRMTexture2::SetColors** method explicitly to change the colors for the Direct3DRMTexture2 object.

See Also **IDirect3DRMTexture2::SetColors**

IDirect3DRMTexture2::GetDecalOrigin

Retrieves the current origin of the decal.

HRESULT GetDecalOrigin(
LONG * *lplX***,**
LONG * *lplY*
);

Parameters *lplX* and *lplY*
Addresses of variables that will be filled with the origin of the decal when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture2::SetDecalOrigin**

IDirect3DRMTexture2::GetDecalScale

Retrieves the scaling property of the given decal.

DWORD GetDecalScale( );

Return Values Returns the scaling property if successful, or -1 otherwise.

See Also **IDirect3DRMTexture2::SetDecalScale**

IDirect3DRMTexture2::GetDecalSize

Retrieves the size of the decal.

HRESULT GetDecalSize(
D3DVALUE **lprvWidth*,*
D3DVALUE **lprvHeight*
);

Parameters *lprvWidth* and *lprvHeight*
Addresses of variables that will be filled with the width and height of the decal when the method returns.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture2::SetDecalSize**

IDirect3DRMTexture2::GetDecalTransparency

Retrieves the transparency property of the decal.

BOOL GetDecalTransparency( );

Return Values Returns TRUE if the decal has a transparent color, FALSE otherwise.

See Also **IDirect3DRMTexture2::SetDecalTransparency**

IDirect3DRMTexture2::GetDecalTransparentColor

Retrieves the transparent color of the decal.

D3DCOLOR GetDecalTransparentColor( );

Return Values Returns the value of the transparent color.

See Also **IDirect3DRMTexture2::SetDecalTransparentColor**

IDirect3DRMTexture2::GetImage

Returns an address of the image that the texture was created with.

D3DRMIMAGE * GetImage( );

Return Values Returns the address of the **D3DRMIMAGE** structure that the current Direct3DRMTexture2 object was created with.

IDirect3DRMTexture2::GetShades

Retrieves the number of shades used for each color in the texture when rendering.

DWORD GetShades( );

Return Values Returns the number of shades.

See Also **IDirect3DRMTexture2::SetShades**

IDirect3DRMTexture2::InitFromFile

Initializes a Direct3DRMTexture2 object using the information in a given file.

HRESULT InitFromFile(
LPCSTR *filename*
);

Parameters

filename
A string that specifyies the file from which the texture initialization information is drawn.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

You must have created the Direct3DRMTexture2 object to be initialized using the **IDirect3DRM2::CreateTexture** or **IDirect3DRM2::CreateTextureFromSurface** methods.

See Also

IDirect3DRMTexture2::InitFromImage, **IDirect3DRMTexture2::InitFromResource2**, **IDirect3DRMTexture2::InitFromSurface**

IDirect3DRMTexture2::InitFromImage

Initializes a texture from an image in memory.

HRESULT InitFromImage(
LPD3DRMIMAGE *lpImage*
);

Parameters

lpImage
Address of a **D3DRMIMAGE** structure describing the source of the texture.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

You must have created the IDirect3DRMTexture2 object to be initialized using the **IDirect3DRM2::CreateTexture** or **IDirect3DRM2::CreateTextureFromSurface** methods.

See Also

IDirect3DRMTexture2::InitFromFile, **IDirect3DRMTexture2::InitFromResource2**, **IDirect3DRMTexture2::InitFromSurface**

IDirect3DRMTexture2::InitFromResource2

Initializes a Direct3DRMTexture2 object from a specified resource.

HRESULT InitFromResource2(
HModule *hModule***,**
LPCTSTR *strName***,**
LPCTSTR *strType*
);

Parameters

hModule
Handle of the specified resource.

strName
The name of the resource to be used to initialize the texture.

strType
The type name of the resource used to initialize the texture. Textures can be stored in RT_BITMAP and RT_RCDATA resource types, or user-defined types. If the resource type is user-defined, this method passes the resource module handle, the resource name, and the resource type to the **FindResource** Win32 API.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

You must have created the IDirect3DRMTexture2 object to be initialized using the **IDirect3DRM2::CreateTexture** or **IDirect3DRM2::CreateTextureFromSurface** methods.

See Also

IDirect3DRMTexture2::InitFromFile, **IDirect3DRMTexture2::InitFromImage**, **IDirect3DRMTexture2::InitFromSurface**

IDirect3DRMTexture2::InitFromSurface

Initializes a Direct3DRMTexture2 object by using the data from a given DirectDraw surface. This method performs the same as **IDirect3DRMTexture::InitFromSurface**.

HRESULT InitFromSurface(
LPDIRECTDRAWSURFACE *lpDDS*
);

Parameters *lpDDS*
Address of a DirectDraw surface from which initialization information is drawn.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks You must have created the IDirect3DRMTexture2 object to be initialized using the **IDirect3DRM2::CreateTexture** or **IDirect3DRM2::CreateTextureFromSurface** methods.

See Also **IDirect3DRMTexture2::InitFromFile**, **IDirect3DRMTexture2::InitFromImage**, **IDirect3DRMTexture2::InitFromResource2**

IDirect3DRMTexture2::SetColors

Sets the maximum number of colors used for rendering a Direct3DRMTexture2 object. This method is required only in the ramp color model.

HRESULT SetColors(
DWORD *ulColors*
);

Parameters *ulColors*
Number of colors. The default value is 8.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture2::GetColors**

IDirect3DRMTexture2::SetDecalOrigin

Sets the origin of the decal as an offset from the top left of the decal.

HRESULT SetDecalOrigin(
LONG *lX*,
LONG *lY*
);

Parameters *lX* and *lY*
New origin, in decal coordinates, for the decal. The default origin is [0, 0].

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The decal's origin is mapped to its frame's position when rendering. For example, the origin of a decal of a cross would be set to the middle of the decal, and the origin of an arrow pointing down would be set to midway along the bottom edge.

This method is also used to add a decal origin key to a Direct3DRMTextureInterpolator object.

See Also **IDirect3DRMTexture2::GetDecalOrigin**

IDirect3DRMTexture2::SetDecalScale

Sets the scaling property for a decal.

```
HRESULT SetDecalScale(
    DWORD dwScale
    );
```

Parameters *dwScale*
If this parameter is TRUE, depth is taken into account when the decal is scaled. If it is FALSE, depth information is ignored. The default value is TRUE.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture2::GetDecalScale**

IDirect3DRMTexture2::SetDecalSize

Sets the size of the decal to be used if the decal is being scaled according to its depth in the scene.

```
HRESULT SetDecalSize(
    D3DVALUE rvWidth,
    D3DVALUE rvHeight
    );
```

Parameters

rvWidth and *rvHeight*
New width and height, in model coordinates, of the decal. The default size is [1, 1].

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a decal size key to a Direct3DRMTextureInterpolator object.

See Also

IDirect3DRMTexture2::GetDecalSize

IDirect3DRMTexture2::SetDecalTransparency

Sets the transparency property of the decal.

HRESULT SetDecalTransparency(
BOOL *bTransp*
);

Parameters

bTransp
If this parameter is TRUE, the decal has a transparent color. If it is FALSE, it has an opaque color. The default value is FALSE.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRMTexture2::GetDecalTransparency

IDirect3DRMTexture2::SetDecalTransparentColor

Sets the transparent color for a decal.

HRESULT SetDecalTransparentColor(
D3DCOLOR *rcTransp*
);

Parameters

rcTransp
New transparent color. The default transparent color is black.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a decal transparent color key to a Direct3DRMTextureInterpolator object.

See Also **IDirect3DRMTexture2::GetDecalTransparentColor**

IDirect3DRMTexture2::SetShades

Sets the maximum number of shades to use for each color for the Direct3DRMTexture2 object when rendering. This method is required only in the ramp color model.

HRESULT SetShades(
DWORD *ulShades*
);

Parameters *ulShades*
New number of shades. This value must be a power of 2. The default value is 16.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMTexture2::GetShades**

IDirect3DRMUserVisual

Applications use the **IDirect3DRMUserVisual** interface to initialize Direct3DRMUserVisual objects. Note that this initialization is not necessary if the application calls the **IDirect3DRM::CreateUserVisual** method; it is required only if the application calls the **IDirect3DRM::CreateObject** method to create the user-visual object. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMUserVisual Interface*.

The **IDirect3DRMUserVisual** interface supports the **Init** method.

The **IDirect3DRMUserVisual** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMUserVisual** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMUserVisual object is obtained by calling the **IDirect3DRM::CreateUserVisual** method.

IDirect3DRMUserVisual::Init

Initializes a Direct3DRMUserVisual object.

HRESULT Init(
D3DRMUSERVISUALCALLBACK *d3drmUVProc*,
void * *lpArg*
);

Parameters

d3drmUVProc
Application-defined **D3DRMUSERVISUALCALLBACK** callback function.

lpArg
Application-defined data to be passed to the callback function.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Applications can call the **IDirect3DRM::CreateUserVisual** method to create and initialize a user-visual object at the same time. It is necessary to call **IDirect3DRMUserVisual::Init** only when the application has created the user-visual object by calling the **IDirect3DRM::CreateObject** method.

IDirect3DRMViewport

Applications use the methods of the **IDirect3DRMViewport** interface to work with viewport objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMViewport and IDirect3DRMViewportArray Interface*.

The methods of the **IDirect3DRMViewport** interface can be organized into the following groups:

Camera	**GetCamera**
	SetCamera
Clipping planes	**GetBack**
	GetFront
	GetPlane
	SetBack
	SetFront
	SetPlane
Dimensions	**GetHeight**
	GetWidth
Field of view	**GetField**
	SetField
Initialization	**Init**
Miscellaneous	**Clear**
	Configure
	ForceUpdate
	GetDevice
	GetDirect3DViewport
	Pick
	Render
Offsets	**GetX**
	GetY

Projection types	**GetProjection**
	SetProjection
Scaling	**GetUniformScaling**
	SetUniformScaling
Transformations	**InverseTransform**
	Transform

The **IDirect3DRMViewport** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

In addition, the **IDirect3DRMViewport** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMViewport object is obtained by calling the **IDirect3DRM::CreateViewport** method.

IDirect3DRMViewport::Clear

Clears the given viewport to the current background color.

HRESULT Clear( );

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::Configure

Reconfigures the origin and dimensions of a viewport.

HRESULT Configure(
LONG *lX*,
LONG *lY*,
DWORD *dwWidth*,
DWORD *dwHeight*
);

Parameters

lX and *lY*
New position of the viewport.

dwWidth and *dwHeight*
New width and height of the viewport.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method returns D3DRMERR_BADVALUE if *lX* + *dwWidth* or *lY* + *dwHeight* are greater than the width or height of the device, or if any of *lX*, *lY*, *dwWidth*, or *dwHeight* is less than zero.

IDirect3DRMViewport::ForceUpdate

Forces an area of the viewport to be updated. The specified area will be copied to the screen at the next call to the **IDirect3DRMDevice::Update** method.

HRESULT ForceUpdate(
DWORD *dwX1*,
DWORD *dwY1*,
DWORD *dwX2*,
DWORD *dwY2*
);

Parameters

dwX1 and *dwY1*
Upper-left corner of area to be updated.

dwX2 and *dwY2*
Lower-right corner of area to be updated.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The system might update any region that is larger than the specified rectangle, including possibly the entire window.

IDirect3DRMViewport::GetBack

Retrieves the position of the back clipping plane for a viewport.

D3DVALUE GetBack( );

Return Values Returns a value describing the distance between the back clipping plane and the camera.

See Also **IDirect3DRMViewport::SetBack**, *Viewing Frustum*

IDirect3DRMViewport::GetCamera

Retrieves the camera for a viewport.

HRESULT GetCamera(
LPDIRECT3DRMFRAME ***lpCamera***
);

Parameters *lpCamera*
Address of a variable that represents the Direct3DRMFrame object representing the camera.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMViewport::SetCamera**, *Camera*

IDirect3DRMViewport::GetDevice

Retrieves the device associated with a viewport.

HRESULT GetDevice(
LPDIRECT3DRMDEVICE **lpD3DRMDevice*
);

Parameters

lpD3DRMDevice
Address of a variable that represents the Direct3DRMDevice object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::GetDirect3DViewport

Retrieves the Direct3D viewport corresponding to the current Direct3DRMViewport.

HRESULT GetDirect3DViewport(
LPDIRECT3DVIEWPORT * *lplpD3DViewport*
);

Parameters

lplpD3DViewport
Address of a pointer that is initialized with a pointer to the Direct3DViewport object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::GetField

Retrieves the field of view for a viewport.

D3DVALUE GetField( );

Return Values

Returns a value describing the field of view.

See Also **IDirect3DRMViewport::SetField**, *Viewing Frustum*

IDirect3DRMViewport::GetFront

Retrieves the position of the front clipping plane for a viewport.

D3DVALUE GetFront( );

Return Values Returns a value describing the distance from the camera to the front clipping plane.

See Also **IDirect3DRMViewport::SetFront**, *Viewing Frustum*

IDirect3DRMViewport::GetHeight

Retrieves the height, in pixels, of the viewport.

DWORD GetHeight( );

Return Values Returns the pixel height.

IDirect3DRMViewport::GetPlane

Retrieves the dimensions of the viewport on the front clipping plane.

HRESULT GetPlane(
D3DVALUE **lpd3dvLeft*,*
D3DVALUE **lpd3dvRight*,*
D3DVALUE **lpd3dvBottom*,*
D3DVALUE **lpd3dvTop*
);

Parameters *lpd3dvLeft, lpd3dvRight, lpd3dvBottom,* and *lpd3dvTop*
Addresses of variables that will be filled to represent the dimensions of the viewport on the front clipping plane.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMViewport::SetPlane**

IDirect3DRMViewport::GetProjection

Retrieves the projection type for the viewport. A viewport can use either orthographic or perspective projection.

D3DRMPROJECTIONTYPE GetProjection( );

Return Values Returns one of the members of the **D3DRMPROJECTIONTYPE** enumerated type.

See Also **IDirect3DRMViewport::SetProjection**

IDirect3DRMViewport::GetUniformScaling

Retrieves the scaling property used to scale the viewing volume into the larger dimension of the window.

BOOL GetUniformScaling( );

Return Values Returns TRUE if the viewport scales uniformly, or FALSE otherwise.

See Also **IDirect3DRMViewport::SetUniformScaling**

IDirect3DRMViewport::GetWidth

Retrieves the width, in pixels, of the viewport.

DWORD GetWidth( );

Return Values Returns the pixel width.

IDirect3DRMViewport::GetX

Retrieves the x-offset of the start of the viewport on a device.

LONG GetX( );

Return Values Returns the x-offset.

IDirect3DRMViewport::GetY

Retrieves the y-offset of the start of the viewport on a device.

LONG GetY( );

Return Values Returns the y-offset.

IDirect3DRMViewport::Init

Initializes a Direct3DRMViewport object.

HRESULT Init(
LPDIRECT3DRMDEVICE *lpD3DRMDevice*,
LPDIRECT3DRMFRAME *lpD3DRMFrameCamera*,
DWORD *xpos*,
DWORD *ypos*,
DWORD *width*,
DWORD *height*
);

Parameters

lpD3DRMDevice
Address of the DirectD3DRMDevice object associated with this viewport.

lpD3DRMFrameCamera
Address of the camera frame associated with this viewport.

xpos and *ypos*
The x- and y-coordinates of the upper-left corner of the viewport.

width and *height*
Width and height of the viewport.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::InverseTransform

Transforms the vector in the *lprvSrc* parameter in screen coordinates to world coordinates, and returns the result in the *lprvDst* parameter.

HRESULT InverseTransform(
D3DVECTOR * *lprvDst***,**
D3DRMVECTOR4D * *lprvSrc*
);

Parameters *lprvDst*
Address of a **D3DVECTOR** structure that will be filled with the result of the operation when the method returns.

lprvSrc
Address of a **D3DRMVECTOR4D** structure representing the source of the operation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::Pick

Finds a depth-sorted list of objects (and faces, if relevant) that includes the path taken in the hierarchy from the root down to the frame that contained the object.

HRESULT Pick(
LONG *lX*,
LONG *lY*,
LPDIRECT3DRMPICKEDARRAY* *lplpVisuals*
);

Parameters *lX* and *lY*
Coordinates to be used for picking.

lplpVisuals
Address of a pointer to be initialized with a valid pointer to the *IDirect3DRMPickedArray* interface if the call succeeds.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::Render

Renders a frame hierarchy to the given viewport. Only those visuals on the given frame and any frames below it in the hierarchy are rendered.

HRESULT Render(
LPDIRECT3DRMFRAME *lpD3DRMFrame*
);

Parameters *lpD3DRMFrame*
Address of a variable that represents the Direct3DRMFrame object that represents the frame hierarchy to be rendered.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMViewport::SetBack

Sets the position of the back clipping plane for a viewport.

HRESULT SetBack(
D3DVALUE *rvBack*
);

Parameters *rvBack*
Distance between the camera and the back clipping plane.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is also used to add a back clipping plane position key to a Direct3DRMViewportInterpolator object.

See Also **IDirect3DRMViewport::GetBack**, **IDirect3DRMViewport::SetFront**, *Viewing Frustum*

IDirect3DRMViewport::SetCamera

Sets a camera for a viewport.

HRESULT SetCamera(
LPDIRECT3DRMFRAME *lpCamera*
);

Parameters

lpCamera
Address of a variable that represents the Direct3DRMFrame object that represents the camera.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method sets a viewport's position, direction, and orientation to that of the given camera frame. The view is oriented along the positive z-axis of the camera frame, with the up direction being in the direction of the positive y-axis.

See Also

IDirect3DRMViewport::GetCamera, *Camera*

IDirect3DRMViewport::SetField

Sets the field of view for a viewport.

HRESULT SetField(
D3DVALUE *rvField*
);

Parameters

rvField
New field of view. The default value is 0.5. If this value is less than or equal to zero, this method returns the D3DRMERR_BADVALUE error.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

This method is also used to add a field of view key to a Direct3DRMViewportInterpolator object.

See Also

IDirect3DRMViewport::GetField, *Viewing Frustum*

IDirect3DRMViewport::SetFront

Sets the position of the front clipping plane for a viewport.

HRESULT SetFront(
D3DVALUE *rvFront*
);

Parameters

rvFront
Distance from the camera to the front clipping plane.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

The default position is 1.0. If the value passed is less than or equal to zero, this method returns the D3DRMERR_BADVALUE error.

This method is also used to add a front clipping plane position key to a Direct3DRMViewportInterpolator object.

See Also

IDirect3DRMViewport::GetFront, *Viewing Frustum*

IDirect3DRMViewport::SetPlane

Sets the dimensions of the viewport on the front clipping plane, relative to the camera's z-axis.

HRESULT SetPlane(
D3DVALUE *rvLeft***,**
D3DVALUE *rvRight***,**
D3DVALUE *rvBottom***,**
D3DVALUE *rvTop*
);

Parameters

rvLeft, rvRight, rvBottom, and *rvTop*
Minimum x, maximum x, minimum y, and maximum y coordinates of the four sides of the viewport.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks

Unlike the **IDirect3DRMViewport::SetField** method, which specifies a centered proportional viewport, this method allows you to specify a viewport of arbitrary

proportion and position. For example, this method could be used to construct a sheared viewing frustum to implement a right- or left-eye stereo view.

This method is also used to add a plane key to a Direct3DRMViewportInterpolator object.

See Also **IDirect3DRMViewport::GetPlane**, **IDirect3DRMViewport::SetField**

IDirect3DRMViewport::SetProjection

Sets the projection type for a viewport.

HRESULT SetProjection(
D3DRMPROJECTIONTYPE *rptType*
);

Parameters *rptType*
One of the members of the **D3DRMPROJECTIONTYPE** enumerated type.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRMViewport::GetProjection**

IDirect3DRMViewport::SetUniformScaling

Sets the scaling property used to scale the viewing volume into the larger dimension of the window.

HRESULT SetUniformScaling(
BOOL *bScale*
);

Parameters *bScale*
New scaling property. If this parameter is TRUE, the same horizontal and vertical scaling factor is used to scale the viewing volume. Otherwise, different scaling factors are used to scale the viewing volume exactly into the window. The default setting is TRUE.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks This method is typically used with the **IDirect3DRMViewport::SetPlane** method to support banding.

See Also **IDirect3DRMViewport::GetUniformScaling**

IDirect3DRMViewport::Transform

Transforms the vector in the *lprvSrc* parameter in world coordinates to screen coordinates, and returns the result in the *lprvDst* parameter.

HRESULT Transform(
D3DRMVECTOR4D * *lprvDst*,
D3DVECTOR * *lprvSrc*
);

Parameters

lprvDst
Address of a **D3DRMVECTOR4D** structure that acts as the destination for the transformation operation.

lprvSrc
Address of a **D3DVECTOR** structure that acts as the source for the transformation operation.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

Remarks The result of the transformation is a four-element homogeneous vector. The point represented by the resulting vector is visible if the following equations are true:

$$wx_{min} \le x < wx_{max}$$
$$wy_{min} \le y < wy_{max}$$
$$0 \le z < w$$

where

$$x_{min} = viewport_x - viewport_{width} / 2$$
$$x_{max} = viewport_x + viewport_{width} / 2$$
$$y_{min} = viewport_y - viewport_{height} / 2$$
$$y_{max} = viewport_y + viewport_{height} / 2$$

IDirect3DRMWinDevice

Applications use the methods of the **IDirect3DRMWinDevice** interface to respond to window messages in a window procedure. This section is a reference

to the methods of this interface. For a conceptual overview, see *Window Management*.

The **IDirect3DRMWinDevice** interface supports the following methods:

HandleActivate

HandlePaint

The **IDirect3DRMWinDevice** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef

QueryInterface

Release

The Direct3DRMWinDevice object is obtained by calling the **IDirect3DRMObject::QueryInterface** method and specifying IID_IDirect3DRMWinDevice, or by calling a method such as **IDirect3DRM::CreateDeviceFromD3D**. Its methods are inherited from the *IDirect3DRMDevice* interface.

IDirect3DRMWinDevice::HandleActivate

Responds to a Windows WM_ACTIVATE message. This ensures that the colors are correct in the active rendering window.

HRESULT HandleActivate(
WORD *wParam*
);

Parameters

wParam
WPARAM parameter passed to the message-processing procedure with the WM_ACTIVATE message.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMWinDevice::HandlePaint

Responds to a Windows WM_PAINT message. The *hDC* parameter should be taken from the **PAINTSTRUCT** structure given to the Windows **BeginPaint** function. This method should be called before repainting any application areas in the window because it may repaint areas outside the viewports that have been created on the device.

HRESULT HandlePaint(
HDC *hDC*
);

Parameters

hDC
Handle of the device context (DC).

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

IDirect3DRMWrap

Applications use the methods of the **IDirect3DRMWrap** interface to work with wrap objects. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirect3DRMWrap Interface*.

The methods of the **IDirect3DRMWrap** interface can be organized into the following groups:

Initialization	**Init**
Wrap	**Apply**
	ApplyRelative

The **IDirect3DRMWrap** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef
QueryInterface
Release

In addition, the **IDirect3DRMWrap** interface inherits the following methods from the *IDirect3DRMObject* interface:

AddDestroyCallback

Clone

DeleteDestroyCallback

GetAppData

GetClassName

GetName

SetAppData

SetName

The Direct3DRMWrap object is obtained by calling the **IDirect3DRM::CreateWrap** method.

IDirect3DRMWrap::Apply

Applies a Direct3DRMWrap object to its destination object. The destination object is typically a face or a mesh.

HRESULT Apply(
LPDIRECT3DRMOBJECT *lpObject*
);

Parameters

lpObject
Address of the destination object.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRM::CreateWrap

IDirect3DRMWrap::ApplyRelative

Applies the wrap to the vertices of the object, first transforming each vertex by the frame's world transformation and the inverse world transformation of the wrap's reference frame.

HRESULT ApplyRelative(
LPDIRECT3DRMFRAME *frame*,
LPDIRECT3DRMOBJECT *mesh*

```
);
```

Parameters

frame
Direct3DRMFrame object containing the object to wrap.

mesh
Direct3DRMWrap object to apply.

Return Values

Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also

IDirect3DRM::CreateWrap

IDirect3DRMWrap::Init

Initializes a Direct3DRMWrap object.

```
HRESULT Init(
    D3DRMWRAPTYPE d3drmwt,
    LPDIRECT3DRMFRAME lpd3drmfRef,
    D3DVALUE ox,
    D3DVALUE oy,
    D3DVALUE oz,
    D3DVALUE dx,
    D3DVALUE dy,
    D3DVALUE dz,
    D3DVALUE ux,
    D3DVALUE uy,
    D3DVALUE uz,
    D3DVALUE ou,
    D3DVALUE ov,
    D3DVALUE su,
    D3DVALUE sv
    );
```

Parameters

d3drmwt
One of the members of the **D3DRMWRAPTYPE** enumerated type.

lpd3drmfRef
Address of a Direct3DRMFrame object representing the reference frame for this Direct3DRMWrap object.

ox, *oy*, and *oz*
Origin of the wrap.

dx, *dy*, and *dz*
The z-axis of the wrap.

ux, *uy*, and *uz*
The y-axis of the wrap.

ou and *ov*
Origin in the texture.

su and *sv*
Scale factor in the texture.

Return Values Returns D3DRM_OK if successful, or an error otherwise. For a list of possible return codes, see *Direct3D Retained-Mode Return Values*.

See Also **IDirect3DRM::CreateWrap**

Structures

D3DRMBOX

Defines the bounding box retrieved by the **IDirect3DRMMesh::GetBox** and **IDirect3DRMMeshBuilder::GetBox** methods.

```
typedef struct _D3DRMBOX {
    D3DVECTOR min, max;
}D3DRMBOX;
typedef D3DRMBOX *LPD3DRMBOX;
```

Members **min** and **max**
Values defining the bounds of the box. These values are **D3DVECTOR** structures.

See Also **D3DVECTOR**, **IDirect3DRMMesh::GetBox**, **IDirect3DRMMeshBuilder::GetBox**

D3DRMIMAGE

Describes an image that is attached to a texture by the **IDirect3DRM::CreateTexture** method. **IDirect3DRMTexture::GetImage** returns the address of this image.

```
typedef struct _D3DRMIMAGE {
    int                 width, height;
```

```
    int                 aspectx, aspecty;
    int                 depth;
    int                 rgb;
    int                 bytes_per_line;
    void*               buffer1;
    void*               buffer2;
    unsigned long       red_mask;
    unsigned long       green_mask;
    unsigned long       blue_mask;
    unsigned long       alpha_mask;
    int                 palette_size;
    D3DRMPALETTEENTRY*  palette;
}D3DRMIMAGE;
typedef D3DRMIMAGE, *LPD3DRMIMAGE;
```

Members

width and **height**
Width and height of the image, in pixels.

aspectx and **aspecty**
Aspect ratio for nonsquare pixels.

depth
Bits per pixel.

rgb
If this member FALSE, pixels are indices into a palette. Otherwise, pixels encode RGB values.

bytes_per_line
Number of bytes of memory for a scan line. This value must be a multiple of four.

buffer1
Memory to render into (first buffer).

buffer2
Second rendering buffer for double buffering. Set this member to NULL for single buffering.

red_mask, **green_mask**, **blue_mask**, and **alpha_mask**
If **rgb** is TRUE, these members are masks for the red, green, and blue parts of a pixel. Otherwise, they are masks for the significant bits of the red, green, and blue elements in the palette. For example, most SVGA displays use 64 intensities of red, green, and blue, so the masks should all be set to 0xfc.

palette_size
Number of entries in the palette.

palette
If **rgb** is FALSE, this member is the address of a **D3DRMPALETTEENTRY** structure describing the palette entry.

See Also

IDirect3DRM::CreateTexture, **IDirect3DRMTexture::GetImage**

D3DRMLOADMEMORY

Identifies a resource to be loaded when an application calls the **IDirect3DRM::Load** method (or one of the other **Load** methods) and specifies **D3DRMLOAD_FROMMEMORY**.

```
typedef struct _D3DRMLOADMEMORY {
    LPVOID lpMemory;
    DWORD  dSize;
} D3DRMLOADMEMORY, *LPD3DRMLOADMEMORY;
```

Members

lpMemory
Address of a block of memory to be loaded.

dSize
Size, in bytes, of the block of memory to be loaded.

See Also

IDirect3DRM::Load, **IDirect3DRMAnimationSet::Load**, **IDirect3DRMFrame::Load**, **IDirect3DRMMeshBuilder::Load**, **D3DRMLOADOPTIONS**, **D3DRMLOADRESOURCE**

D3DRMLOADRESOURCE

Identifies a resource to be loaded when an application calls the **IDirect3DRM::Load** method (or one of the other **Load** methods) and specifies **D3DRMLOAD_FROMRESOURCE**.

```
typedef struct _D3DRMLOADRESOURCE {
    HMODULE hModule;
    LPCTSTR lpName;
    LPCTSTR lpType;
} D3DRMLOADRESOURCE, *LPD3DRMLOADRESOURCE;
```

Members

hModule
Handle of the module containing the resource to be loaded. If this member is NULL, the resource must be attached to the calling executable file.

lpName
Name of the resource to be loaded. For example, if the resource is a mesh, this member should specify the name of the mesh file.

lpType
User-defined type identifying the resource.

Remarks

If the high-order word of the **lpName** or **lpType** member is zero, the low-order word specifies the integer identifier of the name or type of the given resource.

Otherwise, those parameters are long pointers to null-terminated strings. If the first character of the string is a pound sign (#), the remaining characters represent a decimal number that specifies the integer identifier of the resource's name or type. For example, the string "#258" represents the integer identifier 258. An application should reduce the amount of memory required for the resources by referring to them by integer identifier instead of by name.

When an application calls a **Load** method and specifies D3DRMLOAD_FROMRESOURCE, it does not need to find or unlock any resources; the system handles this automatically.

See Also **IDirect3DRM::Load**, **IDirect3DRMAnimationSet::Load**, **IDirect3DRMFrame::Load**, **IDirect3DRMMeshBuilder::Load**, **D3DRMLOADMEMORY**, **D3DRMLOADOPTIONS**

D3DRMPALETTEENTRY

Describes the color palette used in a **D3DRMIMAGE** structure. This structure is used only if the **rgb** member of the **D3DRMIMAGE** structure is FALSE. (If it is TRUE, RGB values are used.)

```
typedef struct _D3DRMPALETTEENTRY {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
    unsigned char flags;
}D3DRMPALETTEENTRY;
typedef D3DRMPALETTEENTRY, *LPD3DRMPALETTEENTRY;
```

Members

red, **green**, and **blue**
Values defining the primary color components that define the palette. These values can range from 0 through 255.

flags
Value defining how the palette is used by the renderer. This value is one of the members of the **D3DRMPALETTEFLAGS** enumerated type.

See Also **D3DRMIMAGE**, **D3DRMPALETTEFLAGS**

D3DRMPICKDESC

Contains the pick position and face and group identifiers of the objects retrieved by the **IDirect3DRMPickedArray::GetPick** method.

```
typedef struct _D3DRMPICKDESC {
    ULONG       ulFaceIdx;
    LONG        lGroupIdx;
    D3DVECTOR   vPosition;
} D3DRMPICKDESC, *LPD3DRMPICKDESC;
```

Members

ulFaceIdx
Face index of the retrieved object.

lGroupIdx
Group index of the retrieved object.

vPosition
Value describing the position of the retrieved object. This value is a **D3DVECTOR** structure.

See Also **D3DVECTOR**, **IDirect3DRMPickedArray::GetPick**

D3DRMPICKDESC2

Contains the face and group identifiers, pick position, horizontal and vertical texture coordinates for the vertex, vertex normal, and color of the objects retrieved by the **IDirect3DRMPicked2Array::GetPick** method.

```
typedef struct _D3DRMPICKDESC2
{
    ULONG    ulFaceIdx;
    LONG lGroupIdx;
    D3DVECTOR    dvPosition;
    D3DVALUE tu;
    D3DVALUE tv;
    D3DVECTOR    dvNormal;
    D3DCOLOR dcColor;

} D3DRMPICKDESC2, *LPD3DRMPICKDESC2;
```

Members

ulFaceIdx
Face index of the retrieved object.

lGroupIdx
Group index of the retrieved object.

vPosition
Value describing the position of the retrieved object. This value is a **D3DVECTOR** structure.

tu and **tv**
Horizontal and vertical texture coordinates, respectively, for the vertex.

dvNormal
Normal vector for the vertex.

dcColor
Vertex color.

D3DRMPMESHLOADSTATUS

Contains the loading status of a progressive mesh loaded with the **IDirect3DRMProgressiveMesh::Load** method. This structure can be retrieved with the **IDirect3DRMProgressiveMesh::GetLoadStatus** method.

```
typedef struct _D3DRMPMESHLOADSTATUS {
    DWORD dwSize;
    DWORD dwPMeshSize;
    DWORD dwBaseMeshSize;
    DWORD dwBytesLoaded;
    DWORD dwVerticesLoaded;
    DWORD dwFacesLoaded;
    DWORD dwFlags;
} D3DRMPMESHLOADSTATUS;
typedef D3DRMPMESHLOADSTATUS *LPD3DRMPMESHLOADSTATUS;
```

Members

dwSize
Size of the structure.

dwPMeshSize
Total size of the progressive mesh in bytes.

dwBaseMeshSize
Size of the base mesh in bytes.

dwBytesLoaded
Total number of bytes loaded.

dwVerticesLoaded
Number of vertices loaded.

dwFacesLoaded
Number of faces loaded.

dwFlags
Flags that indicate the status of the progressive mesh loading. Can be one of the following values:

D3DRMPMESHSTATUS_VALID – The progressive mesh object contains valid data.

D3DRMPMESHSTATUS_INTERRUPTED – The download was interrupted either because the application called **IDirect3DRMProgressiveMesh::Abort** or because the connection was lost.

D3DRMPMESHSTATUS_BASEMESHCOMPLETE – The base mesh has been downloaded.

D3DRMPMESHSTATUS_COMPLETE – All data has been downloaded.

D3DRMPMESHSTATUS_RENDERABLE – It is now possible to render the mesh.

See Also **IDirect3DRMProgressiveMesh::GetLoadStatus**, **IDirect3DRMProgressiveMesh::Load**

D3DRMQUATERNION

Describes the rotation used by the **IDirect3DRMAnimation::AddRotateKey** method, and the quaternion used in **IDirect3DRMFrame2::SetQuaternion**. It is also used in several of Direct3D's mathematical functions.

```
typedef struct _D3DRMQUATERNION {
    D3DVALUE   s;
    D3DVECTOR  v;
}D3DRMQUATERNION;
typedef D3DRMQUATERNION, *LPD3DRMQUATERNION;
```

See Also **IDirect3DRMAnimation::AddRotateKey**, **IDirect3DRMFrame2::SetQuaternion**, **D3DRMQuaternionFromRotation**, **D3DRMQuaternionMultiply**, **D3DRMQuaternionSlerp**, **D3DRMMatrixFromQuaternion**

D3DRMRAY

Defines the direction and starting position of the ray in **IDirect3DRMFrame2::RayPick**.

```
typedef struct _D3DRMRAY
{   D3DVECTOR dvDir;
    D3DVECTOR dvPos;
} D3DRMRAY, *LPD3DRMRAY;
```

Members

dvDir

The direction of the ray used in a ray pick.

dvPos

The starting position of the ray used in a ray pick.

D3DRMVECTOR4D

Describes the screen coordinates used as the destination of a transformation by the **IDirect3DRMViewport::Transform** method and as the source of a transformation by the **IDirect3DRMViewport::InverseTransform** method.

```
typedef struct _D3DRMVECTOR4D {
    D3DVALUE x;
    D3DVALUE y;
    D3DVALUE z;
    D3DVALUE w;
}D3DRMVECTOR4D;
typedef D3DRMVECTOR4D, *LPD3DRMVECTOR4D;
```

Members

x, **y**, **z**, and **w**
Values of the **D3DVALUE** type describing homogeneous values. These values define the result of the transformation.

See Also

IDirect3DRMViewport::Transform, **IDirect3DRMViewport::InverseTransform**

D3DRMVERTEX

Describes a vertex in a Direct3DRMMesh object.

```
typedef struct _D3DRMVERTEX{
    D3DVECTOR position;
    D3DVECTOR normal;
    D3DVALUE  tu, tv;
    D3DCOLOR  color;
} D3DRMVERTEX;
```

Members

position
Position of the vertex.

normal
Normal vector for the vertex.

tu and **tv**
Horizontal and vertical texture coordinates, respectively, for the vertex.

color
Vertex color.

See Also

IDirect3DRMMesh::GetVertices, **IDirect3DRMMesh::SetVertices**

Enumerated Types

D3DRMCOLORSOURCE

Describes the color source of a Direct3DRMMeshBuilder object. You can set the color source by using the **IDirect3DRMMeshBuilder::SetColorSource** method. To retrieve it, use the **IDirect3DRMMeshBuilder::GetColorSource** method.

```
typedef enum _D3DRMCOLORSOURCE{
    D3DRMCOLOR_FROMFACE,
    D3DRMCOLOR_FROMVERTEX
} D3DRMCOLORSOURCE;
```

Values

D3DRMCOLOR_FROMFACE
The object's color source is a face.

D3DRMCOLOR_FROMVERTEX
The object's color source is a vertex.

See Also

IDirect3DRMMeshBuilder::SetColorSource, **IDirect3DRMMeshBuilder::GetColorSource**

D3DRMCOMBINETYPE

Specifies how to combine two matrices.

```
typedef enum _D3DRMCOMBINETYPE{
    D3DRMCOMBINE_REPLACE,
    D3DRMCOMBINE_BEFORE,
    D3DRMCOMBINE_AFTER
} D3DRMCOMBINETYPE;
```

Values

D3DRMCOMBINE_REPLACE
The supplied matrix replaces the frame's current matrix.

D3DRMCOMBINE_BEFORE
The supplied matrix is multiplied with the frame's current matrix and precedes the current matrix in the calculation.

D3DRMCOMBINE_AFTER
The supplied matrix is multiplied with the frame's current matrix and follows the current matrix in the calculation.

Remarks The order of the supplied and current matrices when they are multiplied together is important because matrix multiplication is not commutative.

See Also **IDirect3DRMFrame::AddRotation**, **IDirect3DRMFrame::AddScale**, **IDirect3DRMFrame::AddTransform**, **IDirect3DRMFrame::AddTranslation**

D3DRMFILLMODE

One of the enumerated types that is used in the definition of the **D3DRMRENDERQUALITY** type.

```
typedef enum _D3DRMFILLMODE {
    D3DRMFILL_POINTS    = 0 * D3DRMLIGHT_MAX,
    D3DRMFILL_WIREFRAME = 1 * D3DRMLIGHT_MAX,
    D3DRMFILL_SOLID     = 2 * D3DRMLIGHT_MAX,
    D3DRMFILL_MASK      = 7 * D3DRMLIGHT_MAX,
    D3DRMFILL_MAX       = 8 * D3DRMLIGHT_MAX
} D3DRMFILLMODE;
```

Values

D3DRMFILL_POINTS
Fills points only; minimum fill mode.

D3DRMFILL_WIREFRAME
Fill wire frames.

D3DRMFILL_SOLID
Fill solid objects.

D3DRMFILL_MASK
Fill using a mask.

D3DRMFILL_MAX
Maximum value for fill mode.

See Also **D3DRMLIGHTMODE**, **D3DRMSHADEMODE**, **D3DRMRENDERQUALITY**

D3DRMFOGMODE

Contains values that specify how rapidly and in what ways the fog effect intensifies with increasing distance from the camera.

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.)

```
typedef enum _D3DRMFOGMODE{
    D3DRMFOG_LINEAR,
    D3DRMFOG_EXPONENTIAL,
    D3DRMFOG_EXPONENTIALSQUARED
} D3DRMFOGMODE;
```

Values

D3DRMFOG_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.

D3DRMFOG_EXPONENTIAL
The fog effect intensifies exponentially, according to the following formula:

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

D3DRMFOG_EXPONENTIALSQUARED
The fog effect intensifies exponentially with the square of the distance, according to the following formula:

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

Remarks

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

You can specify the fog's density and start and end points by using the **IDirect3DRMFrame::SetSceneFogParams** method. In the formulas for the exponential fog modes, *e* is the base of the natural logarithms; its value is approximately 2.71828.

See Also

IDirect3DRMFrame::SetSceneFogMode, **IDirect3DRMFrame::SetSceneFogParams**

D3DRMFRAMECONSTRAINT

Describes the axes of rotation to constrain when viewing a Direct3DRMFrame object. The **IDirect3DRMFrame::LookAt** method uses this enumerated type.

```
typedef enum _D3DRMFRAMECONSTRAINT {
    D3DRMCONSTRAIN_Z,
    D3DRMCONSTRAIN_Y,
    D3DRMCONSTRAIN_X
} D3DRMFRAMECONSTRAINT;
```

Values

D3DRMCONSTRAIN_Z
Use only x and y rotations.

D3DRMCONSTRAIN_Y
Use only x and z rotations.

D3DRMCONSTRAIN_X
Use only y and z rotations.

See Also **IDirect3DRMFrame::LookAt**

D3DRMLIGHTMODE

One of the enumerated types that is used in the definition of the **D3DRMRENDERQUALITY** type.

```
typedef enum _D3DRMLIGHTMODE {
    D3DRMLIGHT_OFF      = 0 * D3DRMSHADE_MAX,
    D3DRMLIGHT_ON       = 1 * D3DRMSHADE_MAX,
    D3DRMLIGHT_MASK     = 7 * D3DRMSHADE_MAX,
    D3DRMLIGHT_MAX      = 8 * D3DRMSHADE_MAX
} D3DRMLIGHTMODE;
```

Values

D3DRMLIGHT_OFF
Lighting is off.

D3DRMLIGHT_ON
Lighting is on.

D3DRMLIGHT_MASK
Lighting uses a mask.

D3DRMLIGHT_MAX
Maximum lighting mode.

See Also **D3DRMFILLMODE**, **D3DRMSHADEMODE**, **D3DRMRENDERQUALITY**

D3DRMLIGHTTYPE

Defines the light type in calls to the **IDirect3DRM::CreateLight** method.

```
typedef enum _D3DRMLIGHTTYPE{
    D3DRMLIGHT_AMBIENT,
    D3DRMLIGHT_POINT,
    D3DRMLIGHT_SPOT,
    D3DRMLIGHT_DIRECTIONAL,
    D3DRMLIGHT_PARALLELPOINT
} D3DRMLIGHTTYPE;
```

Values

D3DRMLIGHT_AMBIENT
Light is an ambient source.

D3DRMLIGHT_POINT
Light is a point source.

D3DRMLIGHT_SPOT
Light is a spotlight source.

D3DRMLIGHT_DIRECTIONAL
Light is a directional source.

D3DRMLIGHT_PARALLELPOINT
Light is a parallel point source.

D3DRMMATERIALMODE

Describes the source of material information for visuals rendered with a frame.

```
typedef enum _D3DRMMATERIALMODE{
    D3DRMMATERIAL_FROMMESH,
    D3DRMMATERIAL_FROMPARENT,
    D3DRMMATERIAL_FROMFRAME
} D3DRMMATERIALMODE;
```

Values

D3DRMMATERIAL_FROMMESH
Material information is retrieved from the visual object (the mesh) itself. This is the default setting.

D3DRMMATERIAL_FROMPARENT
Material information, along with color or texture information, is inherited from the parent frame.

D3DRMMATERIAL_FROMFRAME
Material information is retrieved from the frame, overriding any previous material information that the visual object may have possessed.

See Also

IDirect3DRMFrame::GetMaterialMode, **IDirect3DRMFrame::SetMaterialMode**

D3DRMPALETTEFLAGS

Used to define how a color may be used in the **D3DRMPALETTEENTRY** structure.

```
typedef enum _D3DRMPALETTEFLAGS {
    D3DRMPALETTE_FREE,
    D3DRMPALETTE_READONLY,
    D3DRMPALETTE_RESERVED
} D3DRMPALETTEFLAGS;
```

Values

D3DRMPALETTE_FREE
Renderer may use this entry freely.

D3DRMPALETTE_READONLY
Fixed but may be used by renderer.

D3DRMPALETTE_RESERVED
May not be used by renderer.

See Also

D3DRMPALETTEENTRY

D3DRMPROJECTIONTYPE

Defines the type of projection used in a Direct3DRMViewport object. The **IDirect3DRMViewport::GetProjection** and **IDirect3DRMViewport::SetProjection** methods use this enumerated type. The right-hand types enable right-handed projection.

The axes of the camera (see **IDirect3DRMFrame2::SetAxes**) are used in both left-handed and right-handed projection to determine what direction the camera is facing.

```
typedef enum _D3DRMPROJECTIONTYPE{
    D3DRMPROJECT_PERSPECTIVE,
    D3DRMPROJECT_ORTHOGRAPHIC,
    D3DRMPROJECT_RIGHTHANDPERSPECTIVE,
    D3DRMPROJECT_RIGHTHANDORTHOGRAPHIC
} D3DRMPROJECTIONTYPE;
```

Values

D3DRMPROJECT_PERSPECTIVE
The projection is perspective and left-handed.

D3DRMPROJECT_ORTHOGRAPHIC
The projection is orthographic and left-handed.

D3DRMPROJECT_RIGHTHANDPERSPECTIVE
The projection is perspective and right-handed.

D3DRMPROJECT_RIGHTHANDORTHOGRAPHIC
The projection is orthographic and right-handed.

See Also **IDirect3DRMViewport::GetProjection**, **IDirect3DRMViewport::SetProjection**

D3DRMRENDERQUALITY

Combines descriptions of the shading mode, lighting mode, and filling mode for a Direct3DRMMesh object.

```
typedef enum _D3DRMSHADEMODE {
    D3DRMSHADE_FLAT        = 0,
    D3DRMSHADE_GOURAUD     = 1,
    D3DRMSHADE_PHONG       = 2,
    D3DRMSHADE_MASK        = 7,
    D3DRMSHADE_MAX         = 8
} D3DRMSHADEMODE;
typedef enum _D3DRMLIGHTMODE {
    D3DRMLIGHT_OFF          = 0 * D3DRMSHADE_MAX,
    D3DRMLIGHT_ON           = 1 * D3DRMSHADE_MAX,
    D3DRMLIGHT_MASK         = 7 * D3DRMSHADE_MAX,
    D3DRMLIGHT_MAX          = 8 * D3DRMSHADE_MAX
} D3DRMLIGHTMODE;
typedef enum _D3DRMFILLMODE {
    D3DRMFILL_POINTS        = 0 * D3DRMLIGHT_MAX,
    D3DRMFILL_WIREFRAME     = 1 * D3DRMLIGHT_MAX,
    D3DRMFILL_SOLID         = 2 * D3DRMLIGHT_MAX,
    D3DRMFILL_MASK          = 7 * D3DRMLIGHT_MAX,
    D3DRMFILL_MAX           = 8 * D3DRMLIGHT_MAX
} D3DRMFILLMODE;

typedef DWORD D3DRMRENDERQUALITY;

#define D3DRMRENDER_WIREFRAME
(D3DRMSHADE_FLAT+D3DRMLIGHT_OFF+D3DRMFILL_WIREFRAME)
#define D3DRMRENDER_UNLITFLAT
(D3DRMSHADE_FLAT+D3DRMLIGHT_OFF+D3DRMFILL_SOLID)
#define D3DRMRENDER_FLAT
(D3DRMSHADE_FLAT+D3DRMLIGHT_ON+D3DRMFILL_SOLID)
#define D3DRMRENDER_GOURAUD
(D3DRMSHADE_GOURAUD+D3DRMLIGHT_ON+D3DRMFILL_SOLID)
#define D3DRMRENDER_PHONG
(D3DRMSHADE_PHONG+D3DRMLIGHT_ON+D3DRMFILL_SOLID)
```

Values

D3DRMSHADEMODE, **D3DRMLIGHTMODE**, and **D3DRMFILLMODE**
These enumerated types describe the shade, light, and fill modes for Direct3DRMMesh objects.

D3DRMRENDER_WIREFRAME
Display only the edges.

D3DRMRENDER_UNLITFLAT
Flat shaded without lighting.

D3DRMRENDER_FLAT
Flat shaded.

D3DRMRENDER_GOURAUD
Gouraud shaded.

D3DRMRENDER_PHONG
Phong shaded. Phong shading is not currently supported.

See Also

IDirect3DRMMesh::GetGroupQuality, **IDirect3DRMMesh::SetGroupQuality**,**IDirect3DRMDevice2::SetRenderMode**

D3DRMSHADEMODE

One of the enumerated types that is used in the definition of the **D3DRMRENDERQUALITY** type.

```
typedef enum _D3DRMSHADEMODE {
    D3DRMSHADE_FLAT     = 0,
    D3DRMSHADE_GOURAUD  = 1,
    D3DRMSHADE_PHONG    = 2,
    D3DRMSHADE_MASK     = 7,
    D3DRMSHADE_MAX      = 8
} D3DRMSHADEMODE;
```

See Also

D3DRMFILLMODE, **D3DRMLIGHTMODE**, **D3DRMRENDERQUALITY**

D3DRMSORTMODE

Describes how child frames are sorted in a scene.

```
typedef enum _D3DRMSORTMODE {
    D3DRMSORT_FROMPARENT,
```

```
    D3DRMSORT_NONE,
    D3DRMSORT_FRONTTOBACK,
    D3DRMSORT_BACKTOFRONT
} D3DRMSORTMODE;
```

Values

D3DRMSORT_FROMPARENT
Child frames inherit the sorting order of their parents. This is the default setting.

D3DRMSORT_NONE
Child frames are not sorted.

D3DRMSORT_FRONTTOBACK
Child frames are sorted front-to-back.

D3DRMSORT_BACKTOFRONT
Child frames are sorted back-to-front.

See Also **IDirect3DRMFrame::GetSortMode**, **IDirect3DRMFrame::SetSortMode**

D3DRMTEXTUREQUALITY

Describes how a device interpolates between pixels in a texture and pixels in a viewport. This enumerated type is used by the **IDirect3DRMDevice::SetTextureQuality** and **IDirect3DRMDevice::GetTextureQuality** methods.

```
typedef enum _D3DRMTEXTUREQUALITY{
    D3DRMTEXTURE_NEAREST,
    D3DRMTEXTURE_LINEAR,
    D3DRMTEXTURE_MIPNEAREST,
    D3DRMTEXTURE_MIPLINEAR,
    D3DRMTEXTURE_LINEARMIPNEAREST,
    D3DRMTEXTURE_LINEARMIPLINEAR
} D3DRMTEXTUREQUALITY;
```

Values

D3DRMTEXTURE_NEAREST
Choose the nearest pixel in the texture. Does not support MIP mapping.

D3DRMTEXTURE_LINEAR
Linearly interpolate the four nearest pixels. Does not support MIP mapping.

D3DRMTEXTURE_MIPNEAREST
Similar to D3DRMTEXTURE_NEAREST, but uses the appropriate mipmap instead of the texture. Pixel sampling and MIP mapping are both nearest.

D3DRMTEXTURE_MIPLINEAR
Similar to D3DRMTEXTURE_LINEAR, but uses the appropriate mipmap instead of the texture. Pixel sampling is linear; MIP mapping is nearest.

D3DRMTEXTURE_LINEARMIPNEAREST
Similar to D3DRMTEXTURE_MIPNEAREST, but interpolates between the two nearest mipmaps. Pixel sampling is nearest; MIP mapping is linear.

D3DRMTEXTURE_LINEARMIPLINEAR
Similar to D3DRMTEXTURE_MIPLINEAR, but interpolates between the two nearest mipmaps. Both pixel sampling and MIP mapping are linear.

D3DRMUSERVISUALREASON

Defines the reason the system has called the **D3DRMUSERVISUALCALLBACK** callback function.

```
typedef enum _D3DRMUSERVISUALREASON {
    D3DRMUSERVISUAL_CANSEE,
    D3DRMUSERVISUAL_RENDER
} D3DRMUSERVISUALREASON;
```

Values

D3DRMUSERVISUAL_CANSEE
The callback function should return TRUE if the user-visual object is visible in the viewport.

D3DRMUSERVISUAL_RENDER
The callback function should render the user-visual object.

See Also

D3DRMUSERVISUALCALLBACK

D3DRMWRAPTYPE

Defines the type of Direct3DRMWrap object created by the **IDirect3DRM::CreateWrap** method. You can also use this enumerated type to initialize a Direct3DRMWrap object in a call to the **IDirect3DRMWrap::Init** method.

```
typedef enum _D3DRMWRAPTYPE{
    D3DRMWRAP_FLAT,
    D3DRMWRAP_CYLINDER,
    D3DRMWRAP_SPHERE,
    D3DRMWRAP_CHROME
} D3DRMWRAPTYPE;
```

Values

D3DRMWRAP_FLAT
The wrap is flat.

D3DRMWRAP_CYLINDER
The wrap is cylindrical.

D3DRMWRAP_SPHERE
The wrap is spherical.

D3DRMWRAP_CHROME
The wrap allocates texture coordinates so that the texture appears to be reflected onto the objects.

See Also **IDirect3DRM::CreateWrap**, **IDirect3DRMWrap::Init**, *IDirect3DRMWrap Interface*

D3DRMXOFFORMAT

Defines the file type used by the **IDirect3DRMMeshBuilder::Save** method.

```
typedef enum _D3DRMXOFFORMAT{
    D3DRMXOF_BINARY,
    D3DRMXOF_COMPRESSED,
    D3DRMXOF_TEXT
} D3DRMXOFFORMAT;
```

Values

D3DRMXOF_BINARY
File is in binary format. This is the default setting.

D3DRMXOF_COMPRESSED
Not currently supported.

D3DRMXOF_TEXT
File is in text format.

Remarks The D3DRMXOF_BINARY and D3DRMXOF_TEXT settings are mutually exclusive.

See Also **IDirect3DRMMeshBuilder::Save**

D3DRMZBUFFERMODE

Describes whether z-buffering is enabled.

```
typedef enum _D3DRMZBUFFERMODE {
    D3DRMZBUFFER_FROMPARENT,
    D3DRMZBUFFER_ENABLE,
    D3DRMZBUFFER_DISABLE
} D3DRMZBUFFERMODE;
```

Values

D3DRMZBUFFER_FROMPARENT
The frame inherits the z-buffer setting from its parent frame. This is the default setting.

D3DRMZBUFFER_ENABLE
Z-buffering is enabled.

D3DRMZBUFFER_DISABLE
Z-buffering is disabled.

See Also

IDirect3DRMFrame::GetZbufferMode, **IDirect3DRMFrame::SetZbufferMode**

Other Types

D3DRMANIMATIONOPTIONS

Specifies values used by the **IDirect3DRMAnimation::GetOptions** and **IDirect3DRMAnimation::SetOptions** methods to define how animations are played.

```
typedef DWORD D3DRMANIMATIONOPTIONS;
#define D3DRMANIMATION_CLOSED            0x02L
#define D3DRMANIMATION_LINEARPOSITION    0x04L
#define D3DRMANIMATION_OPEN              0x01L
#define D3DRMANIMATION_POSITION          0x00000020L
#define D3DRMANIMATION_SCALEANDROTATION  0x00000010L
#define D3DRMANIMATION_SPLINEPOSITION    0x08L
```

Parameters

D3DRMANIMATION_CLOSED
The animation plays continually, looping back to the beginning whenever it reaches the end. In a closed animation, the last key in the animation should be a repeat of the first. This repeated key is used to indicate the time difference between the last and first keys in the looping animation.

D3DRMANIMATION_LINEARPOSITION
The animation's position is set linearly.

D3DRMANIMATION_OPEN
The animation plays once and stops.

D3DRMANIMATION_POSITION
The animation's position matrix should overwrite any transformation matrices that could be set by other methods.

D3DRMANIMATION_SCALEANDROTATION
The animation's scale and rotation matrix should overwrite any transformation matrices that could be set by other methods.

D3DRMANIMATION_SPLINEPOSITION
The animation's position is set using splines.

D3DRMCOLORMODEL

Describes the color model implemented by the device. For more information, see the **D3DCOLORMODEL** enumerated type.

```
typedef D3DCOLORMODEL D3DRMCOLORMODEL;
```

See Also **D3DCOLORMODEL**

D3DRMINTERPOLATIONOPTIONS

Defines options for the **IDirect3DRMInterpolator::Interpolate** method. These options modify how the object is loaded.

```
typedef DWORD D3DRMINTERPOLATIONOPTIONS;
#define D3DRMINTERPOLATION_OPEN 0x01L
#define D3DRMINTERPOLATION_CLOSED 0x02L
#define D3DRMINTERPOLATION_NEAREST 0x0100L
#define D3DRMINTERPOLATION_LINEAR 0x04L
#define D3DRMINTERPOLATION_SPLINE 0x08L
#define D3DRMINTERPOLATION_VERTEXCOLOR 0x40L
#define D3DRMINTERPOLATION_SLERPNORMALS 0x80L
```

Parameters

D3DRMINTERPOLATION_OPEN
The first and last keys of each key chain will fix the interpolated values outside of the index span.

D3DRMINTERPOLATION_CLOSED
The interpolation is cyclic. The keys effectively repeat infinitely with a period equal to the index span. For compatibility with animations, any key with an index equal to the end of the span is ignored.

D3DRMINTERPOLATION_NEAREST
Nearest key value is used for in-betweening on each key chain.

D3DRMINTERPOLATION_LINEAR
Linear interpolation between the two nearest keys is used for in-betweening on each key chain.

D3DRMINTERPOLATION_SPLINE
A B-spline blending function on the 4 nearest keys is used for in-betweening on each key chain.

D3DRMINTERPOLATION_VERTEXCOLOR
Specifies that vertex colors should be interpolated. Only affects the interpolation of **IDirect3DRMMesh::SetVertices**.

D3DRMINTERPOLATION_SLERPNORMALS
Specifies that vertex normals should be spherically interpolated (not currently implemented). Only affects the interpolation of **IDirect3DRMMesh::SetVertices**.

D3DRMLOADOPTIONS

Defines options for the **IDirect3DRM::Load**, **IDirect3DRMAnimationSet::Load**, **IDirect3DRMFrame::Load**, **IDirect3DRMMeshBuilder::Load** methods, and **IDirect3DRMProgressiveMesh::Load** methods. These options modify how the object is loaded.

```
typedef DWORD D3DRMLOADOPTIONS;
#define D3DRMLOAD_FROMFILE              0x00L
#define D3DRMLOAD_FROMRESOURCE          0x01L
#define D3DRMLOAD_FROMMEMORY            0x02L
#define D3DRMLOAD_FROMURL               0x08L
#define D3DRMLOAD_BYNAME                0x10L
#define D3DRMLOAD_BYPOSITION            0x20L
#define D3DRMLOAD_BYGUID                0x30L
#define D3DRMLOAD_FIRST                 0x40L
#define D3DRMLOAD_INSTANCEBYREFERENCE   0x100L
#define D3DRMLOAD_INSTANCEBYCOPYING     0x200L
#define D3DRMLOAD_ASYNCHRONOUS          0x400L
```

Parameters

Source flags

D3DRMLOAD_FROMFILE
Load from a file. This is the default setting.

D3DRMLOAD_FROMRESOURCE
Load from a resource. If this flag is specified, the *lpvObjSource* parameter of the calling **Load** method must point to a **D3DRMLOADRESOURCE** structure.

D3DRMLOAD_FROMMEMORY
Load from memory. If this flag is specified, the *lpvObjSource* parameter of the calling **Load** method must point to a **D3DRMLOADMEMORY** structure.

D3DRMLOAD_FROMURL
Load from a URL.

Identifier flags

D3DRMLOAD_BYNAME

Load any object by using a specified name.

D3DRMLOAD_BYPOSITION

Load a stand-alone object based on a given zero-based position (that is, the *n*th object in the file). Stand-alone objects can contain other objects, but are not contained by any other objects.

D3DRMLOAD_BYGUID

Load any object by using a specified globally unique identifier (GUID).

D3DRMLOAD_FIRST

This is the default setting. Load the first stand-alone object of the given type (for example, a mesh if the application calls **IDirect3DRMMeshBuilder::Load**). Stand-alone objects can contain other objects, but are not contained by any other objects.

Instance flags

D3DRMLOAD_INSTANCEBYREFERENCE

Check whether an object already exists with the same name as specified and, if so, use an instance of that object instead of creating a new one.

D3DRMLOAD_INSTANCEBYCOPYING

Check whether an object already exists with the same name as specified and, if so, copy that object.

Source flags

D3DRMLOAD_ASYNCHRONOUS

The **Load** call will return immediately. It is up to the application to use events to find out how the load is progressing. By default, loading is done synchronously and the **Load** call will not return until all data has been loaded or an error occurs.

Remarks

Each of the **Load** methods uses an *lpvObjSource* parameter to specify the source of the object and an *lpvObjID* parameter to identify the object. The system interprets the contents of the *lpvObjSource* parameter based on the choice of source flags, and it interprets the contents of the *lpvObjID* parameter based on the choice of identifier flags.

The instance flags do not change the interpretation of any of the parameters. By using the D3DRMLOAD_INSTANCEBYREFERENCE flag, it is possible for an application to load the same file twice without creating any new objects. If an object does not have a name, setting the D3DRMLOAD_INSTANCEBYREFERENCE flag has the same effect as setting the D3DRMLOAD_INSTANCEBYCOPYING flag; the loader creates each unnamed object as a new one, even if some of the objects are identical.

D3DRMMAPPING

Specifies values used by the **IDirect3DRMMesh::GetGroupMapping** and **IDirect3DRMMesh::SetGroupMapping** methods to define how textures are mapped to a group.

```
typedef DWORD D3DRMMAPPING, D3DRMMAPPINGFLAG;
static const D3DRMMAPPINGFLAG D3DRMMAP_WRAPU = 1;
static const D3DRMMAPPINGFLAG D3DRMMAP_WRAPV = 2;
static const D3DRMMAPPINGFLAG D3DRMMAP_PERSPCORRECT = 4;
```

Parameters

D3DRMMAPPINGFLAG
Type equivalent to **D3DRMMAPPING**.

D3DRMMAP_WRAPU
Texture wraps in the u direction.

D3DRMMAP_WRAPV
Texture wraps in the v direction.

D3DRMMAP_PERSPCORRECT
Texture wrapping is perspective-corrected.

Remarks

The D3DRMMAP_WRAPU and D3DRMMAP_WRAPV flags determine how the rasterizer interprets texture coordinates. The rasterizer always interpolates the shortest distance between texture coordinates; that is, a line. The path taken by this line, and the valid values for the u- and v-coordinates, varies with the use of the wrapping flags. If either or both flags is set, the line can wrap around the texture edge in the u or v direction, as if the texture had a cylindrical or toroidal topology. For more information, see *IDirect3DRMWrap Interface*.

D3DRMMATRIX4D

Expresses a transformation as an array. The organization of the matrix entries is D3DRMMATRIX4D[*row*][*column*].

```
typedef D3DVALUE D3DRMMATRIX4D[4][4];
```

See Also

IDirect3DRMFrame::AddTransform, **IDirect3DRMFrame::GetTransform**

D3DRMSAVEOPTIONS

Defines options for the **IDirect3DRMMeshBuilder::Save** method.

```
typedef DWORD D3DRMSAVEOPTIONS;
#define D3DRMXOFSAVE_NORMALS 1
#define D3DRMXOFSAVE_TEXTURECOORDINATES 2
#define D3DRMXOFSAVE_MATERIALS 4
#define D3DRMXOFSAVE_TEXTURENAMES 8
#define D3DRMXOFSAVE_ALL 15
#define D3DRMXOFSAVE_TEMPLATES 16
#define D3DRMSAVE_TEXTURETOPOLOGY 32
```

Parameters

D3DRMXOFSAVE_NORMALS
Save normal vectors in addition to the basic geometry.

D3DRMXOFSAVE_TEXTURECOORDINATES
Save texture coordinates in addition to the basic geometry.

D3DRMXOFSAVE_MATERIALS
Save materials in addition to the basic geometry.

D3DRMXOFSAVE_TEXTURENAMES
Save texture names in addition to the basic geometry.

D3DRMXOFSAVE_ALL
Save normal vectors, texture coordinates, materials, and texture names in addition to the basic geometry.

D3DRMXOFSAVE_TEMPLATES
Save templates with the file. By default, templates are not saved.

D3DRMSAVE_TEXTURETOPOLOGY
Save the mesh's face wrap values (set by **IDirect3DRMMeshBuilder::SetTextureTopology** or **IDirect3DRMFace::SetTextureTopology**). This flag is not included when you pass D3DRMXOFSAVE_ALL to the **Save** method. You should pass D3DRMSAVE_TEXTURETOPOLOGY|D3DRMXOFSAVE_ALL if you really want to save everything. You only need to pass this flag when you've called **IDirect3DRMMeshBuilder::SetTextureTopology** or **IDirect3DRMFace::SetTextureTopology**) calls and want to preserve the values you set.

Return Values

The methods of the Direct3D Retained-Mode Component Object Model (COM) interfaces can return the following values.

D3DRM_OK

No error.

D3DRMERR_BADALLOC

Out of memory.

D3DRMERR_BADDEVICE

Device is not compatible with renderer.

D3DRMERR_BADFILE

Data file is corrupt.

D3DRMERR_BADMAJORVERSION

Bad DLL major version.

D3DRMERR_BADMINORVERSION

Bad DLL minor version.

D3DRMERR_BADOBJECT

Object expected in argument.

D3DRMERR_BADPMDATA

The data in the X File is corrupted. The conversion to a progressive mesh succeeded but produced an invalid progressive mesh in the X File.

D3DRMERR_BADTYPE

Bad argument type passed.

D3DRMERR_BADVALUE

Bad argument value passed.

D3DRMERR_BOXNOTSET

An attempt was made to access a bounding box (for example, with **IDirect3DRMFrame2::GetBox**) when no bounding box was set on the frame.

D3DRMERR_CONNECTIONLOST

Data connection was lost during a load, clone, or duplicate.

D3DRMERR_FACEUSED

Face already used in a mesh.

D3DRMERR_FILENOTFOUND

File cannot be opened.

D3DRMERR_INVALIDDATA

The method received or accessed data that is invalid.

D3DRMERR_INVALIDOBJECT

The method received a pointer to an object that is invalid.

D3DRMERR_INVALIDPARAMS

One of the parameters passed to the method is invalid.

D3DRMERR_NOTDONEYET

Unimplemented.

D3DRMERR_NOTENOUGHDATA

Not enough data has been loaded to perform the requested operation.

D3DRMERR_NOTFOUND

Object not found in specified place.

D3DRMERR_PENDING

The data required to supply the requested information has not finished loading.

D3DRMERR_REQUESTTOOLARGE

An attempt was made to set a level of detail in a progressive mesh greater than the maximum available.

D3DRMERR_REQUESTTOOSMALL

An attempt was made to set the minimum rendering detail of a progressive mesh smaller than the detail in the base mesh (the minimum for rendering).

D3DRMERR_UNABLETOEXECUTE

Unable to carry out procedure.

CHAPTER 5

DirectX File Format

Copyright Notification

Introduction to File Formats

This document specifies the file format introduced with DirectX 2. A binary version of this format was subsequently released with DirectX 3 and this is also detailed in this reference.

The DirectX File Format provides a rich template-driven file format that enables the storage of meshes, textures, animations and user-definable objects. Support

for animation sets allows predefined paths to be stored for playback in real time. Also supported are instancing which allows multiple references to an object, such as a mesh, while storing its data only once per file, and hierarchies to express relationships between data records. The DirectX file format is used natively by the Direct3D Retained Mode API, providing support for reading predefined objects into an application or writing mesh information constructed by the application in real time.

The File Format provides low-level data primitives, upon which applications define higher level primitives via templates. This is the method by which Direct3D defines higher level primitives such as Vectors, Meshes, Matrices and Colors.

This document details the API independent features of the file format and also the method by which Direct3D Retained Mode uses the file format.

File Format Architecture

The DirectX file format is an architecture- and context-free file format. It is template driven and is free of any usage knowledge. The file format may be used by any client application and currently is used by Direct3D Retained Mode to describe geometry data, frame hierarchies and animations.

The rest of this section will deal with the content and syntax of the file format. The file format uses the extension .X when used with the DirectX SDK.

Reserved Words

The following words are reserved and must not be used:

- ARRAY
- BYTE
- CHAR
- CSTRING
- DOUBLE
- DWORD
- FLOAT
- STRING
- TEMPLATE
- UCHAR
- UNICODE
- WORD

Header

The variable length header is compulsory and must be at the beginning of the data stream. The header contains the following

Type	Sub Type	Size	Contents	Content Meaning
Magic Number - required		4 bytes	"xof "	
Version Number - required	Major Number	2 bytes	03	Major version 3
	Minor Number	2 bytes	02	Minor version 2
Format Type - required		4 bytes	"txt "	Text File
			"bin "	Binary File
			"com "	Compressed File
Compression Type - required		4 bytes	"lzw "	
if format type is compressed			"zip "	
			etc...	
Float size - required		4 bytes	0064	64 bit floats
			0032	32 bit floats

Example

```
xof 0302txt 0064
```

Comments

Comments are only applicable in text files. Comments may occur anywhere in the data stream. A comment begins with either C++ style double-slashes "//", or a hash character "#". The comment runs to the next new-line.

```
# This is a comment.
// This is another comment.
```

Templates

Templates define how the data stream is interpreted - the data is modulated by the template definition. A template has the following form:

```
template <template-name> {
<UUID>
<member 1>;
...
<member n>;
[restrictions]
}
```

This section discusses the following parts of a template:

- *Template name*
- *UUID*
- *Members*
- *Restrictions*

Example templates are presented in *Examples*.

Template name

This is an alphanumeric name which may include the underscore character '_'. It must not begin with a digit.

UUID

A universally unique identifier formatted to the OSF DCE standard and surrounded by angle brackets '<' and '>'. For example: <3D82AB43-62DA-11cf-AB39-0020AF71E433>

Members

Template members consist of a named data type followed by an optional name or an array of a named data type. Valid primitive data types are

Type	Size
WORD	16 bits
DWORD	32 bits
FLOAT	IEEE float
DOUBLE	64 bits
CHAR	8 bits
UCHAR	8 bits
BYTE	8 bits
STRING	NULL terminated string
CSTRING	Formatted C-string (currently unsupported)
UNICODE	UNICODE string (currently unsupported)

Additional data types defined by templates encountered earlier in the data stream may also be referenced within a template definition. No forward references are allowed.

Any valid data type can be expressed as an array in the template definition. The basic syntax is as follows

```
array <data-type> <name>[<dimension-size>];
```

Where <dimension-size> can either be an integer or a named reference to another template member whose value is then substituted.

Arrays may be n-dimensional where n is determined by the number of paired square brackets trailing the statement. For example

```
array DWORD FixedHerd[24];
array DWORD Herd[nCows];
array FLOAT Matrix4x4[4][4];
```

Restrictions

Templates may be *open*, *closed* or *restricted*. These restrictions determine which data types may appear in the immediate hierarchy of a data object defined by the template. An open template has no restrictions, a closed template rejects all data types and a restricted template allows a named list of data types. The syntax is as follows

Three periods enclosed by square brackets indicate an open template

```
[ ((( ]
```

A comma separated list of named data types followed optionally by their uuids enclosed by square brackets indicates a restricted template

```
[ { data-type [ UUID ] , }... ]
```

The absence of either of the above indicates a closed template.

Examples

```
template Mesh {
<3D82AB44-62DA-11cf-AB39-0020AF71E433>
DWORD nVertices;
array Vector vertices[nVertices];
DWORD nFaces;
array MeshFace faces[nFaces];
 [ ... ]                // An open template
}
template Vector {
<3D82AB5E-62DA-11cf-AB39-0020AF71E433>
FLOAT x;
FLOAT y;
FLOAT z;
}                        // A closed template
template FileSystem {
<UUID>
STRING name;
[ Directory <UUID>, File <UUID> ]    // A restricted template
}
```

There is one special template - the *Header* template. It is recommended that each application define such a template and use it to define application specific information such as version information. If present, this header will be read by the File Format API and if a *flags* member is available, it will be used to determine how the following data is interpreted. The *flags* member, if defined, should be a DWORD. One bit is currently defined - bit 0. If this is clear, the following data in the file is binary, if set, the following data is text. Multiple header data objects can be used to switch between binary and text during the file.

Data

Data objects contain the actual data or a reference to that data. Each has a corresponding template that specifies the data type.

Data objects have the following form

```
<Identifier> [name] {
<member 1>;
...
<member n>;
}
```

This section discusses the following parts of data objects:

- *Identifier*
- *Name*
- *Members*

Example templates are presented in *Examples*.

Identifier

This is compulsory and must match a previously defined data type or primitive.

Name

This is optional. (See above for syntax definition.)

Members

Data members can be one of the following

Data object

A nested data object. This allows the hierarchical nature of the file format to be expressed. The types of nested data objects allowed in the hierarchy may be restricted. See *Templates* above.

Data reference

A reference to a previously encountered data object. The syntax is as follows

```
{ name }
```

Integer list

A semicolon separated list of integers. For example

```
1; 2; 3;
```

Float list

A semicolon separated list of floats. For example

```
1.0; 2.0; 3.0;
```

String list

A semicolon separated list of strings. For example

```
"Moose"; "Goats"; "Sheep";
```

Use of commas and semicolons

This is perhaps the most complex syntax issue in the file format, but is very strict: Commas are used to separate array members; semicolons terminate every data item.

For example, if we have a template defined as

```
template foo {
DWORD bar;
}
```

then an instance of this would look like

```
foo dataFoo {
1;
}
```

Next, we have a template containing another template

```
template foo {
DWORD bar;
DWORD bar2;
}
template container {
FLOAT aFloat;
foo aFoo;
}
```

then an instance of this would look like

```
container dataContainer {
1.1;
2; 3;;
}
```

Note that the second line that represents the **foo** inside container has two semi-colons at the end of the line. The first indicates the end of the data item **aFoo** (inside container), and the second indicates the end of the **container**.

Next we consider arrays.

```
Template foo {
array DWORD bar[3];
}
```

then an instance of this would look like

```
foo aFoo {
1, 2, 3;
}
```

In the array case, there is no need for the data items to be separated by a semi-colon as they are delineated by a comma. The semi-colon at the end marks the end of the array.

Now consider a template that contains an array of data items defined by a template

```
template foo {
DWORD bar;
DWORD bar2;
}
template container {
DWORD count;
array foo fooArray[count];
}
```

then an instance of this would look like

```
container aContainer {
3;
1;2;,3;4;,5;6;;
}
```

Appendix A: Templates

This appendix lists the templates used by Direct3D's Retained Mode. A familiarity with Direct3D Retained mode data types is assumed.

- *Template Name: Header*
- *Template Name: Vector*
- *Template Name: Coords2d*
- *Template Name: Quaternion*
- *Template Name: Matrix4x4*
- *Template Name: ColorRGBA*
- *Template Name: ColorRGB*
- *Template Name: Indexed Color*
- *Template Name: Boolean*
- *Template Name: Boolean2d*
- *Template Name: Material*
- *Template Name: TextureFilename*
- *Template Name: MeshFace*
- *Template Name: MeshFaceWraps*
- *Template Name: MeshTextureCoords*
- *Template Name: MeshNormals*
- *Template Name: MeshVertexColors*
- *Template Name: MeshMaterialList*
- *Template Name: Mesh*
- *Template Name: FrameTransformMatrix*
- *Template Name: Frame*
- *Template Name: FloatKeys*
- *Template Name: TimedFloatKeys*
- *Template Name: AnimationKey*
- *Template Name: AnimationOptions*
- *Template Name: Animation*
- *Template Name: AnimationSet*

Template Name: Header

UUID

```
<3D82AB43-62DA-11cf-AB39-0020AF71E433>
```

Member Name	Type	Optional Array Size	Optional Data Objects
major	WORD		None
minor	WORD		
flags	DWORD		

Description

This template defines the application specific header for the Direct3D Retained mode usage of the DirectX File Format. The retained mode uses the major and minor flags to specify the current major and minor versions for the retained mode file format.

Template Name: Vector

UUID

<3D82AB5E-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
x	FLOAT		None
y	FLOAT		
z	FLOAT		

Description

This template defines a vector.

Template Name: Coords2d

UUID

<F6F23F44-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
u	FLOAT		None
v	FLOAT		

Description

A two dimensional vector used to define a mesh's texture coordinates.

Template Name: Quaternion

UUID

<10DD46A3-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
s	FLOAT		None
v	Vector		

Description

Currently unused.

Template Name: Matrix4x4

UUID

<F6F23F45-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
matrix	array FLOAT	16	None

Description

This template defines a 4 by 4 matrix. This is used as a frame transformation matrix.

Template Name: ColorRGBA

UUID

<35FF44E0-6C7C-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
red	FLOAT		None
green	FLOAT		
blue	FLOAT		
alpha	FLOAT		

Description

This template defines a color object with an alpha component. This is used for the face color in the material template definition.

Template Name: ColorRGB

UUID

<D3E16E81-7835-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
red	FLOAT		None
green	FLOAT		
blue	FLOAT		

Description

This template defines the basic RGB color object.

Template Name: Indexed Color

UUID

<1630B820-7842-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size
index	DWORD	
ColorRGBA	indexColor	

Description

This template consists of an index parameter and a RGBA color and is used in for defining mesh vertex colors. The index defines the vertex to which the color is applied.

Template Name: Boolean

UUID

<4885AE61-78E8-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
WORD	truefalse		None

Description

Defines a simple boolean type. Should be set to 0 or 1.

Template Name: Boolean2d

UUID

<4885AE63-78E8-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
u	Boolean		None
v	Boolean		

Description

This defines a set of 2 boolean values used in the MeshFaceWraps template in order to define the texture topology of an individual face.

Template Name: Material

UUID

<3D82AB4D-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
faceColor	ColorRGBA		Any
power	FLOAT		
specularColor	ColorRGB		
emissiveColor	ColorRGB		

Description

This template defines a basic material color which can be applied to either a complete mesh or a mesh's individual faces. The power is the specular exponent of the material. Note that the ambient color requires an alpha component.

TextureFilename is an optional data object used by Direct3DRM. If this is not present, the face is untextured.

Template Name: TextureFilename

UUID

<A42790E1-7810-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
filename	STRING		None

Description

This template allows you to specify the filename of a texture to apply to a mesh or a face. This should appear within a material object.

Template Name: MeshFace

UUID

<3D82AB5F-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
nFaceVertexIndices	DWORD		None
faceVertexIndices	array DWORD	nFaceVertexIndices	

Description

This template is used by the Mesh template to define a mesh's faces. Each element of the nFaceVertexIndices array references a mesh vertex used to build the face.

Template Name: MeshFaceWraps

UUID

<4885AE62-78E8-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nFaceWrapValues	DWORD		None
faceWrapValues	Boolean2d		

Description

This template is used to define the texture topology of each face in a wrap. nFaceWrapValues should be equal to the number of faces in a mesh.

Template Name: MeshTextureCoords

UUID

<F6F23F40-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nTextureCoords	DWORD		None
textureCoords	array Coords2d	nTextureCoords	

Description

This template defines a mesh's texture coordinates.

Template Name: MeshNormals

UUID

<F6F23F43-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nNormals	DWORD		None
normals	array Vector	nNormals	
nFaceNormals	DWORD		
faceNormals	array MeshFace	nFaceNormals	

Description

This template defines normals for a mesh. The first array of vectors are the normal vectors themselves, and the second array is an array of indexes specifying which normals should be applied to a given face. nFaceNormals should be equal to the number of faces in a mesh.

Template Name: MeshVertexColors

UUID

<1630B821-7842-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nVertexColors	DWORD		None
vertexColors	array IndexedColor	nVertexColors	

Description

This template specifies vertex colors for a mesh, as opposed to applying a material per face or per mesh.

Template Name: MeshMaterialList

UUID

<F6F23F42-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nMaterials	DWORD		Material
nFaceIndexes	DWORD		
FaceIndexes	array DWORD	nFaceIndexes	

Description

This template is used in a mesh object to specify which material applies to which faces. nMaterials specifies how many materials are present, and materials specify which material to apply.

Template Name: Mesh

UUID

<3D82AB44-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
nVertices	DWORD		Any
vertices	array Vector	nVertices	
nFaces	DWORD		
faces	array MeshFace	nFaces	

Description

This template defines a simple mesh. The first array is a list of vertices and the second array defines the faces of the mesh by indexing into the vertex array.

Optional Data Elements

The following optional data elements are used by Direct3DRM.

MeshFaceWraps	If this is not present, wrapping for both u and v defaults to false.
MeshTextureCoords	If this is not present, there are no texture coordinates.
MeshNormals	If this is not present, normals are generated using the GenerateNormals() member of the API.

MeshVertexColors	If this is not present, the colors default to white.
MeshMaterialList	If this is not present, the material defaults to white.

Template Name: FrameTransformMatrix

UUID

<F6F23F41-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
frameMatrix	Matrix4x4		None

Description

This template defines a local transform for a frame (and all its child objects).

Template Name: Frame

UUID

<3D82AB46-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
none			Any

Description

This template defines a frame. Currently the frame can contain objects of the type Mesh and a FrameTransformMatrix.

Optional Data Elements

The following optional data elements are used by Direct3DRM.

FrameTransformMatrix	If this is not present, no local transform is applied to the frame.
Mesh	Any number of mesh objects that become children of the frame. These can be specified inlined or by reference.

Template Name: FloatKeys

UUID

<10DD46A9-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nValues	DWORD		None
values	array FLOAT	nValues	

Description

This template defines an array of floats and the number of floats in that array. This is used for defining sets of animation keys.

Template Name: TimedFloatKeys

UUID

<F406B180-7B3B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
time	DWORD		None
tfkeys	FloatKeys		

Description

This template defines a set of floats and a positive time used in animations.

Template Name: AnimationKey

UUID

<10DD46A8-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
keyType	DWORD		None
nKeys	DWORD		
keys	array TimedFloatKeys	nKeys	

Description

This template defines a set of animation keys. The keyType parameter specifies whether the keys are rotation, scale or position keys (using the integers 0, 1 or 2 respectively).

Template Name: AnimationOptions

UUID

<E2BF56C0-840F-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
openclosed	DWORD		None
positionquality	DWORD		

Description

This template allows you to set the D3DRM Animation options. The openclosed parameter can be either 0 for a closed or 1 for an open animation. The positionquality parameter is used to set the position quality for any position keys specified and can either be 0 for spline positions or 1 for linear positions. By default an animation is open and uses linear position keys.

Template Name: Animation

UUID

<3D82AB4F-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
none			any

Description

This template contains animations referencing a previous frame. It should contain one reference to a frame and at least one set of AnimationKeys. It can also contain an AnimationOptions data object.

Optional Data Elements

The following optional data elements are used by Direct3DRM.

AnimationKey	An animation is meaningless without AnimationKeys.
AnimationOptions	If this is not present, then an animation is open and uses linear position keys.

Template Name: AnimationSet

UUID

<3D82AB50-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
none			Animation

Description

An AnimationSet contains one or more Animation objects and is the equivalent to the D3D Retained Mode concept of Animation Sets. This means each animation within an animation set has the same time at any given point. Increasing the animation set's time will increase the time for all the animations it contains.

Appendix B: Example

In this appendix we will describe a simple cube and then show how to add textures, frames, and animations to it.

- *A Simple Cube*
- *Adding Textures*
- *Frames and Animations*

A Simple Cube

This file defines a simple cube that has four red and two green sides. Notice in this file that optional information is being used to add information to the data object defined by the Mesh template.

```
Material RedMaterial {
1.000000;0.000000;0.000000;1.000000;;    // R = 1.0, G = 0.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
Material GreenMaterial {
0.000000;1.000000;0.000000;1.000000;;     // R = 0.0, G = 1.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
// Define a mesh with 8 vertices and 12 faces (triangles). Use
// optional data objects in the mesh to specify materials, normals
// and texture coordinates.
Mesh CubeMesh {
8;                                  // 8 vertices
1.000000;1.000000;-1.000000;,     // vertex 0
-1.000000;1.000000;-1.000000;,    // vertex 1
```

```
-1.000000;1.000000;1.000000;,     // etc...
1.000000;1.000000;1.000000;,
1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;1.000000;,
1.000000;-1.000000;1.000000;;

12;                       // 12 faces
3;0,1,2;,                 // face 0 has 3 vertices
3;0,2,3;,                 // etc...
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
3;4,7,6;,
3;4,6,5;;

// All required data has been defined. Now define optional data
// using the hierarchical nature of the file format.
MeshMaterialList {
2;                      // Number of materials used
12;                     // A material for each face
0,                      // face 0 uses the first
0,                      // material
0,
0,
0,
0,
0,
0,
1,                      // face 8 uses the second
1,                      // material
1,
1;;
{RedMaterial}           // References to the definitions
{GreenMaterial}         // of material 0 and 1
}
MeshNormals {
8;                      // define 8 normals
0.333333;0.666667;-0.666667;,
-0.816497;0.408248;-0.408248;,
-0.333333;0.666667;0.666667;,
0.816497;0.408248;0.408248;,
0.666667;-0.666667;-0.333333;,
-0.408248;-0.408248;-0.816497;,
-0.666667;-0.666667;0.333333;,
```

```
0.408248;-0.408248;0.816497;;
12;                    // For the 12 faces,
3;0,1,2;,              // define the normals
3;0,2,3;,
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
3;4,7,6;,
3;4,6,5;;
}
MeshTextureCoords {
8;                          // Define texture coords
0.000000;1.000000;          // for each of the vertices
1.000000;1.000000;
0.000000;1.000000;
1.000000;1.000000;
0.000000;0.000000;
1.000000;0.000000;
0.000000;0.000000;
1.000000;0.000000;;
}
}
```

Adding Textures

In order to add textures, we make use of the hierarchical nature of the file format and add an optional **TextureFilename** data object to the **Material** data objects. So, the Material objects now read as follows:

```
Material RedMaterial {
1.000000;0.000000;0.000000;1.000000;;    // R = 1.0, G = 0.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
TextureFilename {
"tex1.ppm";
}
}
Material GreenMaterial {
0.000000;1.000000;0.000000;1.000000;;     // R = 0.0, G = 1.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
TextureFilename {
```

```
"win95.ppm";
}
}
```

Frames and Animations

This section shows how to add frames and animations to a simple cube.

- *Frames*
- *AnimationSets and Animations*

Frames

A frame is expected to take the following structure

```
Frame Aframe {        // The frame name is chosen for convenience.
FrameTransformMatrix {
...transform data...
}
[ Meshes ] and/or [ More frames]
}
```

So, what we're going to do is place the cube mesh we defined earlier inside a frame with an identity transform. We're then going to apply an animation to this frame.

```
Frame CubeFrame {
FrameTransformMatrix {
1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000,
0.000000, 0.000000, 0.000000, 1.000000;;
}
{CubeMesh}        // We could have the mesh inline, but we'll
                  // use an object reference instead.
}
```

AnimationSets and Animations

Animations and AnimationSets in the file format map directly to Direct3D Retained Mode's animation concepts.

```
Animation Animation0 {        // The name is chosen for convenience.
{ Frame that it applies to - normally a reference }
AnimationKey {
...animation key data...
}
{ ...more animation keys... }
}
```

Animations are then grouped into AnimationSets: AnimationSet AnimationSet0 { // The name is chosen for convenience. { an animation - could be inline or a reference } { ... more animations ... } }

So, what we'll do now is take the cube through an animation

```
AnimationSet AnimationSet0 {
Animation Animation0 {
{CubeFrame}    // Use the frame containing the cube
AnimationKey {
2;             // Position keys
9;             // 9 keys
10; 3; -100.000000, 0.000000, 0.000000;;,
20; 3; -75.000000, 0.000000, 0.000000;;,
30; 3; -50.000000, 0.000000, 0.000000;;,
40; 3; -25.500000, 0.000000, 0.000000;;,
50; 3; 0.000000, 0.000000, 0.000000;;,
60; 3; 25.500000, 0.000000, 0.000000;;,
70; 3; 50.000000, 0.000000, 0.000000;;,
80; 3; 75.500000, 0.000000, 0.000000;;,
90; 3; 100.000000, 0.000000, 0.000000;;;
}
}
}
```

Appendix C: Binary Format

This section details the binary version of the DirectX File Format as introduced with the release of DirectX 3. This appendix should be read in conjunction with the section *File Format Architecture*.

The binary format is a tokenized representation of the text format. Tokens may be stand-alone or accompanied by primitive data records. Stand-alone tokens give grammatical structure and record-bearing tokens supply the necessary data.

Note that all data is stored in little endian format.

A valid binary data stream consists of a header followed by templates and/or data objects.

This section discusses the following components of the binary file format:

- *Header*
- *Templates*
- *Data*
- *Tokens*
- *Token Records*

In addition, example templates are provided, in *Example Templates*. A binary data object is shown in *Example Data*.

Header

The following definitions should be used when reading and writing the binary header directly. Note that compressed data streams are not currently supported and are therefore not detailed here.

```
#define XOFFILE_FORMAT_MAGIC \
  ((long)'x' + ((long)'o' << 8) + ((long)'f' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_VERSION \
  ((long)'0' + ((long)'3' << 8) + ((long)'0' << 16) + ((long)'2' << 24))

#define XOFFILE_FORMAT_BINARY \
  ((long)'b' + ((long)'i' << 8) + ((long)'n' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_TEXT   \
  ((long)'t' + ((long)'x' << 8) + ((long)'t' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_COMPRESSED \
  ((long)'c' + ((long)'m' << 8) + ((long)'p' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_FLOAT_BITS_32 \
  ((long)'0' + ((long)'0' << 8) + ((long)'3' << 16) + ((long)'2' << 24))

#define XOFFILE_FORMAT_FLOAT_BITS_64 \
  ((long)'0' + ((long)'0' << 8) + ((long)'6' << 16) + ((long)'4' << 24))
```

Templates

A template has the following syntax definition

```
template              : TOKEN_TEMPLATE name TOKEN_OBRACE
                            class_id
                            template_parts
                            TOKEN_CBRACE

template_parts        : template_members_part TOKEN_OBRACKET
                        template_option_info
                        TOKEN_CBRACKET
                      | template_members_list

template_members_part : /* Empty */
                      | template_members_list

template_option_info  : ellipsis
```

```
                        | template_option_list

template_members_list :    template_members
                        | template_members_list template_members

template_members      : primitive
                        | array
                        | template_reference

primitive             : primitive_type optional_name TOKEN_SEMICOLON

array                 : TOKEN_ARRAY array_data_type name dimension_list
                          TOKEN_SEMICOLON

template_reference    : name optional_name YT_SEMICOLON

primitive_type        : TOKEN_WORD
                        | TOKEN_DWORD
                        | TOKEN_FLOAT
                        | TOKEN_DOUBLE
                        | TOKEN_CHAR
                        | TOKEN_UCHAR
                        | TOKEN_SWORD
                        | TOKEN_SDWORD
                        | TOKEN_LPSTR
                        | TOKEN_UNICODE
                        | TOKEN_CSTRING

array_data_type       : primitive_type
                        | name

dimension_list        : dimension
                        | dimension_list dimension

dimension             : TOKEN_OBRACKET dimension_size TOKEN_CBRACKET

dimension_size        : TOKEN_INTEGER
                        | name

template_option_list  : template_option_part
                        | template_option_list template_option_part

template_option_part  : name optional_class_id

name                  : TOKEN_NAME

optional_name         : /* Empty */
                        | name

class_id              : TOKEN_GUID
```

```
optional_class_id     : /* Empty */
                      | class_id

ellipsis              : TOKEN_DOT TOKEN_DOT TOKEN_DOT
```

Data

A data object has the following syntax definition

```
object                : identifier optional_name TOKEN_OBRACE
                            optional_class_id
                            data_parts_list
                            TOKEN_CBRACE
data_parts_list       : data_part
                      | data_parts_list data_part

data_part             : data_reference
                      | object
                      | number_list
                      | float_list
                      | string_list

number_list           : TOKEN_INTEGER_LIST

float_list            : TOKEN_FLOAT_LIST

string_list           : string_list_1 list_separator

string_list_1         : string
                      | string_list_1 list_separator string

list_separator        : comma
                      | semicolon

string                : TOKEN_STRING

identifier            : name
                      | primitive_type

data_reference        : TOKEN_OBRACE name optional_class_id TOKEN_CBRACE
```

Tokens

Tokens are written as little endian DWORDs. A list of token values follows. The list is divided into record-bearing and stand-alone tokens.

Record-bearing

```
#define TOKEN_NAME 1
#define TOKEN_STRING 2
#define TOKEN_INTEGER 3
#define TOKEN_GUID 5
#define TOKEN_INTEGER_LIST 6
#define TOKEN_REALNUM_LIST 7
```

Stand-alone

```
#define TOKEN_OBRACE 10
#define TOKEN_CBRACE 11
#define TOKEN_OPAREN 12
#define TOKEN_CPAREN 13
#define TOKEN_OBRACKET 14
#define TOKEN_CBRACKET 15
#define TOKEN_OANGLE 16
#define TOKEN_CANGLE 17
#define TOKEN_DOT 18
#define TOKEN_COMMA 19
#define TOKEN_SEMICOLON 20
#define TOKEN_TEMPLATE 31
#define TOKEN_WORD 40
#define TOKEN_DWORD 41
#define TOKEN_FLOAT 42
#define TOKEN_DOUBLE 43
#define TOKEN_CHAR 44
#define TOKEN_UCHAR 45
#define TOKEN_SWORD 46
#define TOKEN_SDWORD 47
#define TOKEN_VOID 48
#define TOKEN_LPSTR 49
#define TOKEN_UNICODE 50
#define TOKEN_CSTRING 51
#define TOKEN_ARRAY 52
```

Token Records

This section describes the format of the records for each of the record-bearing tokens.

TOKEN_NAME

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_NAME
count	DWORD	4	Length of name field in bytes
name	BYTE array	count	ASCII name

TOKEN_NAME is a variable length record. The token is followed by a *count* value which specifies the number of bytes which follow in the *name* field. An ASCII name of length *count* completes the record.

TOKEN_STRING

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_STRING
count	DWORD	4	Length of string field in bytes
string	BYTE array	count	ASCII string
terminator	DWORD	4	TOKEN_SEMICOLON or TOKEN_COMMA

TOKEN_STRING is a variable length record. The token is followed by a *count* value which specifies the number of bytes which follow in the *string* field. An ASCII string of length *count* continues the record which is completed by a terminating token. The choice of terminator is determined by syntax issues discussed elsewhere.

TOKEN_INTEGER

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_INTEGER
value	DWORD	4	Single integer

TOKEN_INTEGER is a fixed length record. The token is followed by the integer value required.

TOKEN_GUID

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_GUID
data1	DWORD	4	uuid data field 1
data2	WORD	2	uuid data field 2
data3	WORD	2	uuid data field 3
data4	BYTE array	8	uuid data field 4

TOKEN_GUID is a fixed length record. The token is followed by the four data fields as defined by the OSF DCE standard.

TOKEN_INTEGER_LIST

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_INTEGER_LIST

count	DWORD	4	Number of integers in list field
list	DWORD array	4 x count	Integer list

TOKEN_INTEGER_LIST is a variable length record. The token is followed by a count value which specifies the number of integers which follow in the *list* field. For efficiency, consecutive integer lists should be compounded into a single list.

TOKEN_REALNUM_LIST

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_REALNUM_LIST
count	DWORD	4	Number of floats or doubles in list field
list	float/double array	4 or 8 x count	Float or double list

TOKEN_REALNUM_LIST is a variable length record. The token is followed by a count value which specifies the number of floats or doubles which follow in the *list* field. The size of the floating point value (float or double) is determined by the value of *float size* specified in the file header discussed elsewhere. For efficiency, consecutive realnum lists should be compounded into a single list.

Example Templates

Two example binary template definitions are given below. Note that data is stored in little endian format, which is not shown in these illustrative examples.

The closed template *RGB* is identified by the uuid {55b6d780-37ec-11d0-ab39-0020af71e433} and has three members *r*, *g*, and *b* each of type *float*

```
TOKEN_TEMPLATE, TOKEN_NAME, 3, 'R', 'G', 'B', TOKEN_OBRACE,
TOKEN_GUID, 55b6d780, 37ec, 11d0, ab, 39, 00, 20, af, 71, e4, 33,
TOKEN_FLOAT, TOKEN_NAME, 1, 'r', TOKEN_SEMICOLON,
TOKEN_FLOAT, TOKEN_NAME, 1, 'g', TOKEN_SEMICOLON,
TOKEN_FLOAT, TOKEN_NAME, 1, 'b', TOKEN_SEMICOLON,
TOKEN_CBRACE
```

The closed template *Matrix4x4* is identified by the uuid {55b6d781-37ec-11d0-ab39-0020af71e433} and has one member, a two-dimensional array named *matrix* of type *float*

```
TOKEN_TEMPLATE, TOKEN_NAME, 9, 'M', 'a', 't', 'r', 'i', 'x', '4', 'x',
'4', TOKEN_OBRACE,
TOKEN_GUID, 55b6d781, 37ec, 11d0, ab, 39, 00, 20, af, 71, e4, 33,
TOKEN_ARRAY, TOKEN_FLOAT, TOKEN_NAME, 6, 'm', 'a', 't', 'r', 'i', 'x',
TOKEN_OBRACKET, TOKEN_INTEGER, 4, TOKEN_CBRACKET,
```

```
TOKEN_OBRACKET, TOKEN_INTEGER, 4, TOKEN_CBRACKET,
TOKEN_CBRACE
```

Example Data

The binary data object below shows an instance of the *RGB* template defined above. The example object is named *blue* and its three members *r*, *g*, and *b* have the values 0.0, 0.0 and 1.0 respectively. Note that data is stored in little endian format which is not shown in this illustrative example.

```
TOKEN_NAME, 3, 'R', 'G', 'B', TOKEN_NAME, 4, 'b', 'l', 'u', 'e',
TOKEN_OBRACE,
TOKEN_FLOAT_LIST, 3, 0.0, 0.0, 1.0, TOKEN_CBRACE
```

Appendix D: The Conv3ds.exe Utility

The Conv3ds.exe utility program converts 3-D models produced by Autodesk 3-D Studio and other modeling packages into the DirectX file format. By default, it produces binary X files with no templates.

Using Conv3ds.exe

You can run Conv3ds.exe with no options, and it will produce an X file containing a hierarchy of frames. For example, consider the command:

```
conv3ds File.3ds
```

This will produce an X File called File.x. You can use **IDirect3DRMFrame::Load** to load the frame.

Conv3ds.exe Optional Arguments

Conv3ds.exe -A

If the 3DS file contains key frame data, you can use the **-A** option to produce an X file that contains an animation set. The command for this would be:

```
conv3ds -A File.3ds
```

The File.3ds parameter is the name of the file to be converted. You can use **IDirect3DRMAnimationSet::Load** to load the animation.

Conv3ds.exe -m

Use the **-m** option to make an X file that contains a single mesh made from all the objects in the 3DS file.

```
conv3ds -m File.3ds
```

Use **IDirect3DRMMeshBuilder::Load** to load the mesh.

Conv3ds.exe -T

Use the**-T** option to wrap all the objects and frame hierarchies in a single top level frame. Using this option, all the frames and objects in the 3DS file can be loaded with a single call to **IDirect3DRMFrame::Load**. The first top level frame hierarchy in the X file will be loaded. The frame containing all the other frames and meshes is called "x3ds_filename" (without the .3ds extension). The **-T** option will have no effect if it is used with the **-m** option.

Conv3ds.exe -s

The **-s** option allows you to specify a scale factor for all the objects converted in the 3DS file. For example, the following command makes all objects 10 times bigger:

```
conv3ds -s10 File.3ds
```

The following command makes all objects 10 times smaller:

```
conv3ds -s0.1 File.3ds
```

Conv3ds.exe -r

The **-r** option reverses the winding order of the faces when the 3DS file is converted. If, after converting the 3DS file and viewing it in Direct3D, the object appears "inside-out" try converting it with the **-r** option. All Lightwave models exported as 3DS files need this option. See *3DS Files Produced from Lightwave Objects* for details.

Conv3ds.exe -v

The **-v** option turns on verbose output mode. Specify an integer with it. The integers that are currently supported are:

Option	Meaning
-v0	Default. Verbose mode off.
-v1	Prints warnings about bad objects, and prints general information about what the converter is doing.
-v2	Prints basic keyframe information, the objects being included in the conversion process, and information about the objects being saved.
-v3	Very verbose. Mostly useful for the debugging information it provides.

Conv3ds.exe -e

The -e option allows you to change the extension for texture map files. For example, consider the command:

```
conv3ds -e"ppm" File.3ds
```

If File.3ds contains objects that reference the texture map file brick.gif, the X file will reference the texture map file brick.ppm. The converter does not convert the texture map file. The texture map files must be in the D3DPATH when the resulting X File is loaded.

Conv3ds.exe -x

The **-x** option forces **Conv3ds** to produce a text X file, instead of a binary X file. Text files are larger but can be hand edited easily.

Conv3ds.exe -X

The **-X** option forces **Conv3ds** to include the Direct3DRM X File templates in the file. By default the templates are not included.

Conv3ds.exe -t

The **-t** option specifies that the X File produced will not contain texture information.

Conv3ds.exe -N

The **-N** option specifies that the X file produced will not contain normal vector information. All the Direct3DRM Load calls will generate normal vectors for objects with no normal vectors in the X file.

Conv3ds.exe -c

The **-c** option specifies that the X file produced should not contain texture coordinates. By default, if you use the **-m** option, the mesh that is output will contain (0,0) uv texture coordinates if the 3DS object had no texture coordinates.

Conv3ds.exe -f

The **-f** option specifies that the X file produced should not contain a Frame transformation matrix.

Conv3ds.exe - z and **Conv3ds.exe -Z**

The **-z** and **-Z** options allow you to adjust the alpha face color value of all the materials referenced by objects in the X File. For example, the following command causes **Conv3ds.exe** to add 0.1 to all alpha values under 0.2:

```
conv3ds -z0.1 -Z0.2 File.3ds
```

The following command causes **Conv3ds.exe** to subtract 0.2 from the alpha values for all alphas:

```
conv3ds-z"-0.2" -z1 File.3ds
```

Conv3ds.exe -o

The **-o** option allows you to specify the filename for the .X File produced.

Conv3ds.exe -h

The **-h** option tells the converter not to try to resolve any hierarchy information in the 3DS file (usually produced by the keyframer). Instead, all the objects are output in top level frames if **-m** option is not used.

3DS Files Produced from Lightwave Objects

There are several issues with 3DS files exported by the Trans3d plug-in for Lightwave. These are best handled using the following Conv3ds.exe command:

```
conv3ds -r -N -f -h -T|m trans3dfile.3ds
```

All the 3DS objects produced by Trans3d and the Lightwave object editor need their winding order reversed. Otherwise, they appear "inside-out" when displayed. They contain no surface normal vector information.

Hints and Tips

If you can't see objects produced by Conv3ds.exe after loading them into the Direct3DRM viewer, use the scale option **-s** with a scale factor of approximately 100. This will increase scale of the objects in the X file.

If, after loading the object into the viewer and switching from flat shading into gourard shading the object turns dark grey, try converting with the **-N** option.

If the textures aren't loaded after the object is converted, ensure that the object is referencing either .ppm or .bmp files by using the **-e** option. Also ensure that the textures' widths and heights are a power of 2. Make sure the textures are stored in one of the directories in your D3DPATH.

Currently, Conv3ds.exe can't handle dummy frames used in 3DS animations. It ignores them. However, it will convert any child objects.