Resources Overview

Article • 02/08/2021

A Direct2D resource is an object that is used for drawing and is represented by a Direct2D interface, such as **ID2D1Geometry** or **ID2D1RenderTarget**. This topic describes the kinds of Direct2D resources and how they can be shared.

This topic contains the following sections:

About Direct2D Resources

Many hardware-accelerated 2D APIs are designed around a CPU-focused resource model and a set of rendering operations that work well on CPUs.Then, some part of the API is hardware-accelerated. Implementing these APIs requires a resource manager for mapping CPU resources to resources on the GPU. Because of GPU limitations, some operations might be unable to be accelerated in every circumstance. In these cases, the resource manager must communicate back and forth between the CPU and GPU (which is expensive) so that it can transition to CPU rendering. In some cases, it might unpredictably force rendering to fall back completely to the CPU. In addition, rendering operations that appear simple might require temporary intermediate rendering steps that are not exposed in the API and that require additional GPU resources.

Direct2D provides a more direct mapping to making full use of the GPU. It provides two categories of resources: device-independent and device-dependent.

- Device-independent resources, such as **ID2D1Geometry**, are kept on the CPU.
- Device-dependent resources, such as **ID2D1RenderTarget** and **ID2D1LinearGradientBrush**, directly map to resources on the GPU (when hardware acceleration is available). Rendering calls are performed by combining vertex and coverage information from a geometry with texturing information produced by the device-dependent resources.

When you create device-dependent resources, system resources (the GPU, if available, or the CPU) are allocated when the device is created and do not change from one rendering operation to another. This situation eliminates the need for a resource manager. In addition to the general performance improvements that are provided by the elimination of a resource manager, this model enables you to directly control any intermediate rendering.

Because Direct2D provides so much control over resources, you must understand the different kinds of resources and when they can be used together.

Device-Independent Resources

As described in the previous section, device-independent resources always reside on the CPU and are never associated with a hardware rendering device. The following are device-independent resources:

- ID2D1DrawingStateBlock
- ID2D1Factory
- ID2D1Geometry and the interfaces that inherit from it.
- ID2D1GeometrySink and ID2D1SimplifiedGeometrySink
- ID2D1StrokeStyle

Use an ID2D1Factory, itself a device-independent resource, to create device-independent resources. (To create a factory, use the CreateFactory function.)

Except for render targets, all the resources created by a factory are device-independent. A render target is a device-dependent resource.

Device-Dependent Resources

Any resource that is not named in the previous list is a device-dependent resource. Device-dependent resources are associated with a particular rendering device. When hardware acceleration is available, that device is the GPU. In other cases, it is the CPU.

To create most device-dependent resources, use a render target. In most cases, use a factory to create a render target.

The following are examples of device-dependent resources:

- ID2D1Brush and the interfaces that inherit from it. Use a render target to create brushes.
- ID2D1Layer. Use a render target to create layers.

- ID2D1RenderTarget and the interfaces that inherit from it. To create a render target, use a factory or another render target.

> ⓘ Note
>
> Starting with Windows 8, there are new interfaces that create device dependent resources. An ID2D1Device and a ID2D1DeviceContext can share a resource if the device context and the resource are created from the same **ID2D1Device**.

Device-dependent resources become unusable when the associated rendering devices become unavailable. This means that when you receive the D2DERR_RECREATE_TARGET error for a render target, you must re-create the render target and all its resources.

Sharing Factory Resources

You can share all device-independent resources created by a factory with all other resources (device-independent or device-dependent) created by the same factory. For example, you can use two ID2D1RenderTarget objects to draw the same ID2D1RectangleGeometry if both of those **ID2D1RenderTarget** objects were created by the same factory.

The sink interfaces (ID2D1SimplifiedGeometrySink, ID2D1GeometrySink, and ID2D1TessellationSink) may be shared with resources created by any factory. Unlike other interfaces in Direct2D, any implementation of a sink interface can be used. For example, you could use your own implementation of **ID2D1SimplifiedGeometrySink**.

Sharing Render Target Resources

Your ability to share resources created by a render target depends on the kind of render target. When you create a render target of type D2D1_RENDER_TARGET_TYPE_DEFAULT, the resources created by that render target can only be used by that render target (unless the render target fits into one of the categories described in the following sections). This occurs because you do not know what device the render target will end up using—it could end up rendering to local hardware, software, or to a remote client's hardware. For example, you could write a program that stops working when it is displayed remotely or when the render target is increased in size beyond the maximum size supported by the rendering hardware.

The following sections describe the circumstances under which a resource created by one render target can be shared with another render target.

Hardware Render Targets

You can share resources between any render target that explicitly uses hardware, as long as the remoting mode is compatible. The remoting mode is only guaranteed to be compatible when both render targets use the D2D1_RENDER_TARGET_USAGE_FORCE_BITMAP_REMOTING or **D2D1_RENDER_TARGET_USAGE_GDI_COMPATIBLE** usage flag, or if neither flag is specified. These settings guarantee that the resources will always be located on the same computer. To specify a usage mode, set the **usage** field of the D2D1_RENDER_TARGET_PROPERTIES structure that you used to create the render target with one or more **D2D1_RENDER_TARGET_USAGE** flags.

To create a render target that explicitly uses hardware rendering, set the **type** field of the D2D1_RENDER_TARGET_PROPERTIES structure that you used to create the render target to D2D1_RENDER_TARGET_TYPE_HARDWARE.

DXGI Surface Render Targets

You can share resources created by a DXGI surface render target with any other DXGI surface render target that is using the same underlying Direct3D device.

Compatible Render Targets and Shared Bitmaps

You can share resources between a render target and compatible render targets that are created by that render target. To create a compatible render target, use the ID2D1RenderTarget::CreateCompatibleRenderTarget method.

You can use the ID2D1RenderTarget::CreateSharedBitmap method to create an ID2D1Bitmap that can be shared between the two render targets that are specified in the method call, if the method succeeds. This method will succeed as long as the two render targets use the same underlying device for rendering.

Feedback