Direct3D 12

provides a lower level of hardware abstraction than ever before, which allows developers to significantly improve the multi-thread scaling and CPU utilization of their titles. With Direct3D 12, titles are responsible for their memory management. In addition, by using Direct3D 12, games and titles benefit from reduced GPU overhead via features such as command queues and lists, descriptor tables, and concise pipeline state objects.

Text conventions:

- Colors for functions, methods, structures and interfaces, enumerations and constants, additional notes.
- Bitwiseable types provides values
- CX Structures having a D3DX12 class helper
- p## See page ## for type reference.
- N.A. Not applicable/available to Direct3D 12

Specification and additional resources at: *msdn.microsoft.com/en-us/library/windows/desktop/dn90382*
D3DX12 header available under MIT license at: *github.com/Microsoft/DirectX-Graphics-Samples*

Return Codes

Returns codes are reported via **HRESULT** values, defined in winerror.h

Success codes

Success codes (status codes) are non-negative.

```
S_OK
S_FALSE
DXGI_STATUS_OCCLUDED
DXGI_STATUS_CLIPPED
DXGI_STATUS_NO_REDIRECTION
DXGI_STATUS_NO_DESKTOP_ACCESS
DXGI_STATUS_GRAPHICS_VIDPN_SOURCE_IN_USE
DXGI_STATUS_MODE_CHANGED
DXGI_STATUS_MODE_CHANGE_IN_PROGRESS
```

Error codes

Error codes are negative.

```
E_NOTIMPL
E_UNEXPECTED
E_OUTOFMEMORY
E_INVALIDARG
DXGI_ERROR_INVALID_CALL
DXGI_ERROR_NOT_FOUND
DXGI_ERROR_MORE_DATA
DXGI_ERROR_UNSUPPORTED
DXGI_ERROR_DEVICE_REMOVED
DXGI_ERROR_DEVICE_HUNG
DXGI_ERROR_DEVICE_RESET
DXGI_ERROR_WAS_STILL_DRAWING
DXGI_ERROR_FRAME_STATISTICS_DISJOINT
DXGI_ERROR_GRAPHICS_VIDPN_SOURCE_IN_USE
DXGI_ERROR_DRIVER_INTERNAL_ERROR
DXGI_ERROR_NONEXCLUSIVE
DXGI_ERROR_NOT_CURRENTLY_AVAILABLE
DXGI_ERROR_REMOTE_CLIENT_DISCONNECTED
DXGI_ERROR_REMOTE_OUTOFMEMORY
DXGI_ERROR_ACCESS_LOST
DXGI_ERROR_WAIT_TIMEOUT
DXGI_ERROR_SESSION_DISCONNECTED
DXGI_ERROR_RESTRICT_TO_OUTPUT_STALE
DXGI_ERROR_CANNOT_PROTECT_CONTENT
DXGI_ERROR_ACCESS_DENIED
DXGI_ERROR_NAME_ALREADY_EXISTS
DXGI_ERROR_SDK_COMPONENT_MISSING
DXGI_ERROR_NOT_CURRENT
DXGI_ERROR_HW_PROTECTION_OUTOFMEMORY
D3D12_ERROR_ADAPTER_NOT_FOUND
D3D12_ERROR_DRIVER_VERSION_MISMATCH
```

Factory

Everything starts with a factory!

```
HRESULT CreateDXGIFactory(
  REFIID riid,
  void** ppFactory );
ppFactory: a IDXGIFactory
Note: does not work for Store apps
```

```
HRESULT CreateDXGIFactory1(
  REFIID riid,
  void** ppFactory );
ppFactory: a IDXGIFactory1

HRESULT CreateDXGIFactory2(
  UINT Flags,
  REFIID riid,
  void** ppFactory );
Flags: 0, DXGI_CREATE_FACTORY_DEBUG
ppFactory: a IDXGIFactory2

UINT IDXGIFactory3::GetCreationFlags();
Returns: 0, DXGI_CREATE_FACTORY_DEBUG
```

Adapter

Represents a physical display subsystem (including one or more GPUs, DACs and video memory).

Adapter enumeration

```
HRESULT IDXGIFactory::EnumAdapters(
  UINT Adapter,
  IDXGIAdapter** ppAdapter );

HRESULT IDXGIFactory1::EnumAdapters1(
  UINT Adapter,
  IDXGIAdapter1** ppAdapter );

BOOL IDXGIFactory1::IsCurrent();

HRESULT IDXGIFactory4::EnumWarpAdapter(
  REFIID riid,
  void** ppvAdapter );
ppvAdapter: a IDXGIAdapter

struct LUID{
  DWORD LowPart;
  LONG HighPart; };

LUID ID3D12Device::GetAdapterLuid();

HRESULT IDXGIFactory4::EnumAdapterByLuid(
  LUID AdapterLuid,
  REFIID riid,
  void** ppvAdapter );
ppvAdapter: a IDXGIAdapter
```

Adapter description

```
struct DXGI_ADAPTER_DESC{
  WCHAR Description[128];
  UINT VendorId;
  UINT DeviceId;
  UINT SubSysId;
  UINT Revision;
  SIZE_T DedicatedVideoMemory;
  SIZE_T DedicatedSystemMemory;
  SIZE_T SharedSystemMemory;
  LUID AdapterLuid; };
```

```
HRESULT IDXGIAdapter::GetDesc(
  DXGI_ADAPTER_DESC* pDesc );

enum DXGI_ADAPTER_FLAG
DXGI_ADAPTER_FLAG_X where X is
  NONE,
  REMOTE,
  SOFTWARE,
  FORCE_DWORD N.A.

struct DXGI_ADAPTER_DESC1{
  ...
  UINT Flags; };
... like DXGI_ADAPTER_DESC
Flags: a DXGI_ADAPTER_FLAG value

HRESULT IDXGIAdapter1::GetDesc1(
  DXGI_ADAPTER_DESC1* pDesc );

enum DXGI_GRAPHICS_PREEMPTION_GRANULARITY
DXGI_GRAPHICS_PREEMPTION_X where X is
  DMA_BUFFER_BOUNDARY,
  PRIMITIVE_BOUNDARY,
  TRIANGLE_BOUNDARY,
  PIXEL_BOUNDARY,
  INSTRUCTION_BOUNDARY

enum DXGI_COMPUTE_PREEMPTION_GRANULARITY
DXGI_COMPUTE_PREEMPTION_X where X is
  DMA_BUFFER_BOUNDARY,
  DISPATCH_BOUNDARY,
  THREAD_GROUP_BOUNDARY,
  THREAD_BOUNDARY,
  INSTRUCTION_BOUNDARY

struct DXGI_ADAPTER_DESC2{
  ...
  DXGI_GRAPHICS_PREEMPTION_GRANULARITY
    GraphicsPreemptionGranularity;
  DXGI_COMPUTE_PREEMPTION_GRANULARITY
    ComputePreemptionGranularity; };
... like DXGI_ADAPTER_DESC1

HRESULT IDXGIAdapter2::GetDesc2(
  DXGI_ADAPTER_DESC2* pDesc );
```

Device

Represents a virtual adapter: it is used to create command allocators, command lists, command queues, fences, resources, pipeline state objects, heaps, root signatures, samplers, and resource views.

Device creation

```
HRESULT WINAPI D3D12CreateDevice(
  IUnknown* pAdapter,
  D3D_FEATURE_LEVEL MinimumFeatureLevel, p19
  REFIID riid,
  void** ppDevice );
ppDevice: a ID3D12Device
```

Retrieve device

```
HRESULT ID3D12DeviceChild::GetDevice(
  REFIID riid,
  void** ppvDevice );
ppDevice: a ID3D12Device
```

Device removed reason

```
HRESULT ID3D12Device::GetDeviceRemovedReason();
```

Device capabilities

Applications need to discover device capabilities not described by the device creation **D3D_FEATURE_LEVEL** p19.

```
enum D3D12_SHADER_MIN_PRECISION_SUPPORT
D3D12_SHADER_MIN_PRECISION_SUPPORT_X where X is
  NONE = 0,
  10_BIT = 0x1,
  16_BIT = 0x2

enum D3D12_TILED_RESOURCES_TIER
D3D12_TILED_RESOURCES_X where X is
  TIER_NOT_SUPPORTED,
  TIER_1,
  TIER_2,
  TIER_3

enum D3D12_RESOURCE_BINDING_TIER
D3D12_RESOURCE_BINDING_X where X is
  TIER_1,
  TIER_2,
  TIER_3
```

Where tiers indicates the flexibility in the amount of resources available to the pipeline. Bold entries highlight improvements over the previous tier:

Binding Tier (min feature level)	Tier 1 (11.0+)	Tier 2 (11.0+)	Tier 3 (11.1+)
Max # descriptors in a shader visible CBV/SRV/UAV heap	1000000	1000000	1000000+
Max CBVs in all descriptor tables per shader stage	14	14	**full heap**
Max SRVs in all descriptor tables per shader stage	128	**full heap**	full heap
Max UAVs in all descriptor tables across all stages	8 (FL 11.0) 64 (FL 11.1+)	**64**	**full heap**
Max Samplers in all descriptor tables per shader stage	16	**full heap**	full heap

```
enum D3D12_CONSERVATIVE_RASTERIZATION_TIER
D3D12_CONSERVATIVE_RASTERIZATION_X where X is
  TIER_NOT_SUPPORTED,
  TIER_1,
  TIER_2,
  TIER_3

enum D3D12_CROSS_NODE_SHARING_TIER
D3D12_CROSS_NODE_SHARING_X where X is
  TIER_NOT_SUPPORTED,
  TIER_1_EMULATED,
  TIER_1,
  TIER_2

enum D3D12_RESOURCE_HEAP_TIER
D3D12_RESOURCE_HEAP_X where X is
  TIER_1,
  TIER_2

struct D3D12_FEATURE_DATA_D3D12_OPTIONS{
  BOOL DoublePrecisionFloatShaderOps;
  BOOL OutputMergerLogicOp;
  D3D12_SHADER_MIN_PRECISION_SUPPORT MinPrecisionSupport;
  D3D12_TILED_RESOURCES_TIER TiledResourcesTier;
  D3D12_RESOURCE_BINDING_TIER ResourceBindingTier;
  BOOL PSSpecifiedStencilRefSupported;
  BOOL TypedUAVLoadAdditionalFormats;
  BOOL ROVsSupported;
  D3D12_CONSERVATIVE_RASTERIZATION_TIER ConservativeRasterizationTier;
  UINT MaxGPUVirtualAddressBitsPerResource;
  BOOL StandardSwizzle64KBSupported;
  D3D12_CROSS_NODE_SHARING_TIER CrossNodeSharingTier;
  BOOL CrossAdapterRowMajorTextureSupported;
  BOOL VPAndRTArrayIndexFromAnyShaderFeedingRasterizer-
SupportedWithoutGSEmulation;
  D3D12_RESOURCE_HEAP_TIER ResourceHeapTier; };

struct D3D12_FEATURE_DATA_ARCHITECTURE{
  UINT NodeIndex;
  BOOL TileBasedRenderer;
  BOOL UMA;
  BOOL CacheCoherentUMA; };

struct D3D12_FEATURE_DATA_FEATURE_LEVELS{
  UINT NumFeatureLevels;
  const D3D_FEATURE_LEVEL* pFeatureLevelsRequested; p19
  D3D_FEATURE_LEVEL MaxSupportedFeatureLevel; p19 };

enum D3D12_FORMAT_SUPPORT1
D3D12_FORMAT_SUPPORT1_X where X is
  NONE = 0,
  BUFFER = 0x1,
  IA_VERTEX_BUFFER = 0x2,
  IA_INDEX_BUFFER = 0x4,
  SO_BUFFER = 0x8,
  TEXTURE1D = 0x10,
  TEXTURE2D = 0x20,
  TEXTURE3D = 0x40,
  TEXTURECUBE = 0x80,
  SHADER_LOAD = 0x100,
  SHADER_SAMPLE = 0x200,
  SHADER_SAMPLE_COMPARISON = 0x400,
  SHADER_SAMPLE_MONO_TEXT = 0x800,
  MIP = 0x1000,
  RENDER_TARGET = 0x4000,
  BLENDABLE = 0x8000,
  DEPTH_STENCIL = 0x10000,
  MULTISAMPLE_RESOLVE = 0x40000,
  DISPLAY = 0x80000,
  CAST_WITHIN_BIT_LAYOUT = 0x100000,
  MULTISAMPLE_RENDERTARGET = 0x200000,
  MULTISAMPLE_LOAD = 0x400000,
  SHADER_GATHER = 0x800000,
  BACK_BUFFER_CAST = 0x1000000,
  TYPED_UNORDERED_ACCESS_VIEW = 0x2000000,
  SHADER_GATHER_COMPARISON = 0x4000000,
  DECODER_OUTPUT = 0x8000000,
  VIDEO_PROCESSOR_OUTPUT = 0x10000000,
  VIDEO_PROCESSOR_INPUT = 0x20000000,
  VIDEO_ENCODER = 0x40000000

enum D3D12_FORMAT_SUPPORT2
D3D12_FORMAT_SUPPORT2_X where X is
  NONE = 0,
  UAV_ATOMIC_ADD = 0x1,
  UAV_ATOMIC_BITWISE_OPS = 0x2,
  UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE = 0x4,
  UAV_ATOMIC_EXCHANGE = 0x8,
  UAV_ATOMIC_SIGNED_MIN_OR_MAX = 0x10,
  UAV_ATOMIC_UNSIGNED_MIN_OR_MAX = 0x20,
  UAV_TYPED_LOAD = 0x40,
  UAV_TYPED_STORE = 0x80,
  OUTPUT_MERGER_LOGIC_OP = 0x100,
  TILED = 0x200,
  MULTIPLANE_OVERLAY = 0x4000

struct D3D12_FEATURE_DATA_FORMAT_SUPPORT{
  DXGI_FORMAT Format; p19
  D3D12_FORMAT_SUPPORT1 Support1;
  D3D12_FORMAT_SUPPORT2 Support2; };
```

```
enum D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS
D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_X where X is
  NONE = 0,
  TILED_RESOURCE = 0x1

struct D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS{
  DXGI_FORMAT Format; p19
  UINT SampleCount;
  D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS Flags;
  UINT NumQualityLevels; };

struct D3D12_FEATURE_DATA_FORMAT_INFO{
  DXGI_FORMAT Format; p19
  UINT8 PlaneCount; };

struct D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT{
  UINT MaxGPUVirtualAddressBitsPerResource;
  UINT MaxGPUVirtualAddressBitsPerProcess; };

enum D3D_SHADER_MODEL
D3D_SHADER_X where X is
  MODEL_5_1,
  MODEL_6_0

struct D3D12_FEATURE_DATA_SHADER_MODEL {
  D3D_SHADER_MODEL HighestShaderModel; };

struct D3D12_FEATURE_DATA_D3D12_OPTIONS1 {
  BOOL WaveOps;
  UINT WaveLaneCountMin;
  UINT WaveLaneCountMax;
  UINT TotalLaneCount;
  BOOL ExpandedComputeResourceStates;
  BOOL Int64ShaderOps; };

struct D3D12_FEATURE_DATA_ROOT_SIGNATURE {
  D3D_ROOT_SIGNATURE_VERSION HighestVersion; p19 };
```

```
enum D3D12_FEATURE
D3D12_FEATURE_X where X is
  D3D12_OPTIONS,
  ARCHITECTURE,
  FEATURE_LEVELS,
  FORMAT_SUPPORT,
  MULTISAMPLE_QUALITY_LEVELS,
  FORMAT_INFO,
  GPU_VIRTUAL_ADDRESS_SUPPORT,
  SHADER_MODEL,
  D3D12_OPTIONS1,
  ROOT_SIGNATURE

HRESULT ID3D12Device::CheckFeatureSupport(
  D3D12_FEATURE Feature,
  void* pFeatureSupportData,
  UINT FeatureSupportDataSize );
```
pFeatureSupportData: a capabilities record mapped as follows:

D3D12_FEATURE value	D3D12_FEATURE_DATA
D3D12_OPTIONS	D3D12_FEATURE_DATA_D3D12_OPTIONS
ARCHITECTURE	D3D12_FEATURE_DATA_ARCHITECTURE
FEATURE_LEVELS	D3D12_FEATURE_DATA_FEATURE_LEVELS
FORMAT_SUPPORT	D3D12_FEATURE_DATA_FORMAT_SUPPORT
MULTISAMPLE_QUALITY_LEVELS	D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS
FORMAT_INFO	D3D12_FEATURE_DATA_FORMAT_INFO
GPU_VIRTUAL_ADDRESS_SUPPORT	D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT
SHADER_MODEL	D3D12_FEATURE_DATA_SHADER_MODEL
D3D12_OPTIONS1	D3D12_FEATURE_DATA_D3D12_OPTIONS1
ROOT_SIGNATURE	D3D12_FEATURE_DATA_ROOT_SIGNATURE

```
UINT ID3D12Device::GetNodeCount();
```

Command Queue

Submits and synchronizes command lists, and updates resource tile mappings. There is only one graphics queue per adapter node.

Command queue creation

```
enum D3D12_COMMAND_QUEUE_PRIORITY
D3D12_COMMAND_QUEUE_PRIORITY_X where X is
  NORMAL,
  HIGH

enum D3D12_COMMAND_QUEUE_FLAGS
D3D12_COMMAND_QUEUE_FLAG_X where X is
  NONE = 0,
  DISABLE_GPU_TIMEOUT = 0x1

struct D3D12_COMMAND_QUEUE_DESC{
  D3D12_COMMAND_LIST_TYPE Type; p18
  INT Priority;
  D3D12_COMMAND_QUEUE_FLAGS Flags;
  UINT NodeMask; };
```
Priority: D3D12_COMMAND_QUEUE_PRIORITY

```
HRESULT ID3D12Device::CreateCommandQueue(
  const D3D12_COMMAND_QUEUE_DESC* pDesc,
  REFIID riid,
  void** ppCommandQueue );
```
ppCommandQueue: a ID3D12CommandQueue

```
D3D12_COMMAND_QUEUE_DESC
ID3D12CommandQueue::GetDesc();
```

Queue execution.

```
void ID3D12CommandQueue::ExecuteCommandLists(
  UINT NumCommandLists,
  ID3D12CommandList* const* ppCommandLists );
```

Queue synchronization

```
HRESULT ID3D12CommandQueue::Signal(
  ID3D12Fence* pFence,
  UINT64 Value );

HRESULT ID3D12CommandQueue::Wait(
  ID3D12Fence* pFence,
  UINT64 Value );
```

Command Allocator

Corresponds to the underlying allocations in which GPU commands are stored. Applies to both direct command lists and bundles.

Command allocator creation

```
HRESULT ID3D12Device::CreateCommandAllocator(
  D3D12_COMMAND_LIST_TYPE type, p18
  REFIID riid,
  void** ppCommandAllocator );
```
ppCommandAllocator: a ID3D12CommandAllocator

Command allocator reset

Needed to re-use the memory that is associated with the command allocator. Do not reset while a command list is still being executed, proper command lists synchronization is needed.

```
HRESULT ID3D12CommandAllocator::Reset();
```

Command List

Represents an ordered set of commands that the GPU executes. There are two levels of indirection: direct command lists (corresponding to a command buffer that the GPU can execute) and bundles (executed only directly via a direct command list).

Command list creation

```
HRESULT ID3D12Device::CreateCommandList(
  UINT nodeMask,
  D3D12_COMMAND_LIST_TYPE type, p18
  ID3D12CommandAllocator* pCommandAllocator,
  ID3D12PipelineState* pInitialState,
  REFIID riid,
  void** ppCommandList );
```
ppCommandList: ID3D12CommandList, ID3D12GraphicsCommandList

Reset command list

Needed to re-use the memory that is associated with the command list. A command list can be reset at any time after ID3D12CommandQueue::ExecuteCommandLists is called.

```
HRESULT ID3D12GraphicsCommandList::Reset(
  ID3D12CommandAllocator* pAllocator,
  ID3D12PipelineState* pInitialState );
```

Command list capabilities

```
D3D12_COMMAND_LIST_TYPE ID3D12CommandList::GetType(); p18
```

States set-up

States set-up is needed before recording the related commands.

```
void ID3D12GraphicsCommandList::ClearState(
  ID3D12PipelineState* pPipelineState );

struct D3D12_INDEX_BUFFER_VIEW{
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation; p19
  UINT SizeInBytes;
  DXGI_FORMAT Format; p19 };

void ID3D12GraphicsCommandList::IASetIndexBuffer(
  const D3D12_INDEX_BUFFER_VIEW* pView );

enum D3D12_PRIMITIVE_TOPOLOGY
D3D_PRIMITIVE_TOPOLOGY_X where X is
  UNDEFINED,
  POINTLIST,
  LINELIST,
  LINESTRIP,
  TRIANGLELIST,
  TRIANGLESTRIP,
  LINELIST_ADJ,
  LINESTRIP_ADJ,
  TRIANGLELIST_ADJ,
  TRIANGLESTRIP_ADJ,
  1_CONTROL_POINT_PATCHLIST,
  2_CONTROL_POINT_PATCHLIST,
  3_CONTROL_POINT_PATCHLIST,
  4_CONTROL_POINT_PATCHLIST,
  5_CONTROL_POINT_PATCHLIST,
  6_CONTROL_POINT_PATCHLIST,
  7_CONTROL_POINT_PATCHLIST,
  8_CONTROL_POINT_PATCHLIST,
  9_CONTROL_POINT_PATCHLIST,
  10_CONTROL_POINT_PATCHLIST,
  11_CONTROL_POINT_PATCHLIST,
  12_CONTROL_POINT_PATCHLIST,
  13_CONTROL_POINT_PATCHLIST,
  14_CONTROL_POINT_PATCHLIST,
  15_CONTROL_POINT_PATCHLIST,
  16_CONTROL_POINT_PATCHLIST,
  17_CONTROL_POINT_PATCHLIST,
  18_CONTROL_POINT_PATCHLIST,
  19_CONTROL_POINT_PATCHLIST,
  20_CONTROL_POINT_PATCHLIST,
  21_CONTROL_POINT_PATCHLIST,
  22_CONTROL_POINT_PATCHLIST,
  23_CONTROL_POINT_PATCHLIST,
  24_CONTROL_POINT_PATCHLIST,
  25_CONTROL_POINT_PATCHLIST,
  26_CONTROL_POINT_PATCHLIST,
  27_CONTROL_POINT_PATCHLIST,
  28_CONTROL_POINT_PATCHLIST,
  29_CONTROL_POINT_PATCHLIST,
  30_CONTROL_POINT_PATCHLIST,
  31_CONTROL_POINT_PATCHLIST,
  32_CONTROL_POINT_PATCHLIST,
  ... N.A.

void ID3D12GraphicsCommandList::IASetPrimitiveTopology(
  D3D12_PRIMITIVE_TOPOLOGY PrimitiveTopology );

struct D3D12_VERTEX_BUFFER_VIEW{
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation; p19
  UINT SizeInBytes;
  UINT StrideInBytes; };

void ID3D12GraphicsCommandList::IASetVertexBuffers(
  UINT StartSlot,
  UINT NumViews,
  const D3D12_VERTEX_BUFFER_VIEW* pViews );

void ID3D12GraphicsCommandList::OMSetBlendFactor(
  const FLOAT BlendFactor[4] );

void ID3D12GraphicsCommandList::OMSetRenderTargets(
  UINT NumRenderTargetDescriptors,
  const D3D12_CPU_DESCRIPTOR_HANDLE* pRenderTargetDescriptors, p18
  BOOL RTsSingleHandleToDescriptorRange,
  const D3D12_CPU_DESCRIPTOR_HANDLE* pDepthStencilDescriptor p18 );

void ID3D12GraphicsCommandList::OMSetStencilRef(
  UINT StencilRef );

void ID3D12GraphicsCommandList::RSSetScissorRects(
  UINT NumRects,
  const D3D12_RECT* pRects p19 );

CX struct D3D12_VIEWPORT{
  FLOAT TopLeftX;
  FLOAT TopLeftY;
  FLOAT Width;
  FLOAT Height;
  FLOAT MinDepth;
  FLOAT MaxDepth; };

void ID3D12GraphicsCommandList::RSSetViewports(
  UINT NumViewports,
  const D3D12_VIEWPORT* pViewports );

void ID3D12GraphicsCommandList::SetDescriptorHeaps(
  UINT NumDescriptorHeaps,
  ID3D12DescriptorHeap* const* ppDescriptorHeaps );

void ID3D12GraphicsCommandList::SetComputeRootSignature(
  ID3D12RootSignature* pRootSignature );

void ID3D12GraphicsCommandList::SetGraphicsRootSignature(
  ID3D12RootSignature* pRootSignature );

void ID3D12GraphicsCommandList::SetPipelineState(
  ID3D12PipelineState* pPipelineState );

enum D3D12_PREDICATION_OP
D3D12_PREDICATION_OP_X where X is
  EQUAL_ZERO,
  NOT_EQUAL_ZERO

void ID3D12GraphicsCommandList::SetPredication(
  ID3D12Resource* pBuffer,
  UINT64 AlignedBufferOffset,
  D3D12_PREDICATION_OP Operation );

struct D3D12_STREAM_OUTPUT_BUFFER_VIEW{
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation; p19
  UINT64 SizeInBytes;
  D3D12_GPU_VIRTUAL_ADDRESS BufferFilledSizeLocation; p19 };

void ID3D12GraphicsCommandList::SOSetTargets(
  UINT StartSlot,
  UINT NumViews,
  const D3D12_STREAM_OUTPUT_BUFFER_VIEW* pViews );
```

Record commands

```
enum D3D12_CLEAR_FLAGS
D3D12_CLEAR_FLAG_ X where X is
  DEPTH = 0x1,
  STENCIL = 0x2

void ID3D12GraphicsCommandList::ClearDepthStencilView(
  D3D12_CPU_DESCRIPTOR_HANDLE DepthStencilView, p18
  D3D12_CLEAR_FLAGS ClearFlags,
  FLOAT Depth,
  UINT8 Stencil,
  UINT NumRects,
  const D3D12_RECT* pRects p19 );

void ID3D12GraphicsCommandList::ClearRenderTargetView(
  D3D12_CPU_DESCRIPTOR_HANDLE RenderTargetView, p18
  const FLOAT ColorRGBA[4],
  UINT NumRects,
  const D3D12_RECT* pRects p19 };
```

```
void ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat(
  D3D12_GPU_DESCRIPTOR_HANDLE ViewGPUHandleInCurrentHeap, p19
  D3D12_CPU_DESCRIPTOR_HANDLE ViewCPUHandle, p18
  ID3D12Resource* pResource,
  const FLOAT Values[4],
  UINT NumRects,
  const D3D12_RECT* pRects p19 );

void ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint(
  D3D12_GPU_DESCRIPTOR_HANDLE ViewGPUHandleInCurrentHeap, p19
  D3D12_CPU_DESCRIPTOR_HANDLE ViewCPUHandle, p18
  ID3D12Resource* pResource,
  const UINT Values[4],
  UINT NumRects,
  const D3D12_RECT* pRects p19 );

void ID3D12GraphicsCommandList::Dispatch(
  UINT ThreadGroupCountX,
  UINT ThreadGroupCountY,
  UINT ThreadGroupCountZ );

void ID3D12GraphicsCommandList::ExecuteBundle(
  ID3D12GraphicsCommandList* pCommandList );

void ID3D12GraphicsCommandList::DrawIndexedInstanced(
  UINT IndexCountPerInstance,
  UINT InstanceCount,
  UINT StartIndexLocation,
  INT BaseVertexLocation,
  UINT StartInstanceLocation );

void ID3D12GraphicsCommandList::DrawInstanced(
  UINT VertexCountPerInstance,
  UINT InstanceCount,
  UINT StartVertexLocation,
  UINT StartInstanceLocation );
```

Close command list

A command lists must be closed to transit out of the recording state and for further recording or submitting usage.

```
HRESULT ID3D12GraphicsCommandList::Close();
```

Synchronization

Explicit synchronization is needed for commands execution between queues and frames, as it is needed for resource access and usage.

Fence synchronization

```
enum D3D12_FENCE_FLAGS
D3D12_FENCE_FLAG_X where X is
  NONE = 0,
  SHARED = 0x1,
  SHARED_CROSS_ADAPTER = 0x2

HRESULT ID3D12Device::CreateFence(
  UINT64 InitialValue,
  D3D12_FENCE_FLAGS Flags,
  REFIID riid,
  void** ppFence );
ppFence: a ID3D12Fence

HRESULT ID3D12Fence::Signal(
  UINT64 Value );

UINT64 ID3D12Fence::GetCompletedValue();

HRESULT ID3D12Fence::SetEventOnCompletion(
  UINT64 Value,
  HANDLE hEvent );

enum D3D12_MULTIPLE_FENCE_WAIT_FLAGS
D3D12_MULTIPLE_FENCE_WAIT_FLAG_X where
X is
  NONE = 0,
  ANY = 0x1,
  ALL = 0

HRESULT ID3D12Device1::
SetEventOnMultipleFenceCompletion(
  ID3D12Fence* const* ppFences,
  const UINT64* pFenceValues,
  UINT NumFences,
  D3D12_MULTIPLE_FENCE_WAIT_FLAGS Flags,
  HANDLE hEvent);

HRESULT ID3D12CommandQueue::Signal(
  ID3D12Fence* pFence,
  UINT64 Value );

HRESULT ID3D12CommandQueue::Wait(
  ID3D12Fence* pFence,
  UINT64 Value );
```

Frames synchronization

```
UINT IDXGISwapChain3::
GetCurrentBackBufferIndex();
```

Resource access synchronization

```
struct D3D12_RESOURCE_TRANSITION_BARRIER{
  ID3D12Resource* pResource;
  UINT Subresource;
  D3D12_RESOURCE_STATES StateBefore; p19
  D3D12_RESOURCE_STATES StateAfter; p19};

struct D3D12_RESOURCE_ALIASING_BARRIER{
  ID3D12Resource* pResourceBefore;
  ID3D12Resource* pResourceAfter; };

struct D3D12_RESOURCE_UAV_BARRIER{
  ID3D12Resource* pResource; };

enum D3D12_RESOURCE_BARRIER_TYPE
D3D12_RESOURCE_BARRIER_TYPE_X where X
is
  TRANSITION,
  ALIASING,
  UAV

enum D3D12_RESOURCE_BARRIER_FLAGS
D3D12_RESOURCE_BARRIER_FLAG_X where X
is
  NONE = 0,
  BEGIN_ONLY = 0x1,
  END_ONLY = 0x2

CX struct D3D12_RESOURCE_BARRIER{
  D3D12_RESOURCE_BARRIER_TYPE Type;
  D3D12_RESOURCE_BARRIER_FLAGS Flags;
  union{
    D3D12_RESOURCE_TRANSITION_BARRIER Transition;
    D3D12_RESOURCE_ALIASING_BARRIER Aliasing;
    D3D12_RESOURCE_UAV_BARRIER UAV; }; };

void ID3D12GraphicsCommandList::ResourceBarrier(
  UINT NumBarriers,
  const D3D12_RESOURCE_BARRIER* pBarriers );

struct D3D12_DISCARD_REGION{
  UINT NumRects;
  const D3D12_RECT* pRects; p19
  UINT FirstSubresource;
  UINT NumSubresources; };

void ID3D12GraphicsCommandList::DiscardResource(
  ID3D12Resource* pResource,
  const D3D12_DISCARD_REGION* pRegion );
```

Root Signature

Defines what resources are bound to the graphics/compute pipeline. The maximum size of a root signature is 64 DWORDs. Only one root signature can bound to a given pipeline at a time. It is also possible to specify root signatures in HLSL via Shader Model 5.1.

Root signature creation

```
enum D3D12_ROOT_PARAMETER_TYPE
D3D12_ROOT_PARAMETER_TYPE_X where X is
  DESCRIPTOR_TABLE,
  32BIT_CONSTANTS,
  CBV,
  SRV,
  UAV

enum D3D12_DESCRIPTOR_RANGE_TYPE
D3D12_DESCRIPTOR_RANGE_TYPE_X where X is
  SRV,
  UAV,
  CBV,
  SAMPLER

CX struct D3D12_DESCRIPTOR_RANGE{
  D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
  UINT NumDescriptors;
  UINT BaseShaderRegister;
  UINT RegisterSpace;
  UINT OffsetInDescriptorsFromTableStart; };

CX struct D3D12_ROOT_DESCRIPTOR_TABLE{
  UINT NumDescriptorRanges;
  const D3D12_DESCRIPTOR_RANGE* pDescriptorRanges; };
```

```
CX struct D3D12_ROOT_CONSTANTS{
  UINT ShaderRegister;
  UINT RegisterSpace;
  UINT Num32BitValues; };

CX struct D3D12_ROOT_DESCRIPTOR{
  UINT ShaderRegister;
  UINT RegisterSpace; };

enum D3D12_SHADER_VISIBILITY
D3D12_SHADER_VISIBILITY_X where X is
  ALL,
  VERTEX,
  HULL,
  DOMAIN,
  GEOMETRY,
  PIXEL

CX struct D3D12_ROOT_PARAMETER{
  D3D12_ROOT_PARAMETER_TYPE ParameterType;
  union{
    D3D12_ROOT_DESCRIPTOR_TABLE DescriptorTable;
    D3D12_ROOT_CONSTANTS Constants;
    D3D12_ROOT_DESCRIPTOR Descriptor; };
  D3D12_SHADER_VISIBILITY ShaderVisibility; };

enum D3D12_STATIC_BORDER_COLOR
D3D12_STATIC_BORDER_COLOR_X where X is
  TRANSPARENT_BLACK,
  OPAQUE_BLACK,
  OPAQUE_WHITE

CX struct D3D12_STATIC_SAMPLER_DESC{
  D3D12_FILTER Filter; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressU; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressV; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressW; p19
  FLOAT MipLODBias;
  UINT MaxAnisotropy;
  D3D12_COMPARISON_FUNC ComparisonFunc; p18
  D3D12_STATIC_BORDER_COLOR BorderColor;
  FLOAT MinLOD;
  FLOAT MaxLOD;
  UINT ShaderRegister;
  UINT RegisterSpace;
  D3D12_SHADER_VISIBILITY ShaderVisibility; };

enum D3D12_ROOT_SIGNATURE_FLAGS
D3D12_ROOT_SIGNATURE_FLAG_X where X is
  NONE= 0,
  ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT = 0x1,
  DENY_VERTEX_SHADER_ROOT_ACCESS = 0x2,
  DENY_HULL_SHADER_ROOT_ACCESS = 0x4,
  DENY_DOMAIN_SHADER_ROOT_ACCESS = 0x8,
  DENY_GEOMETRY_SHADER_ROOT_ACCESS = 0x10,
  DENY_PIXEL_SHADER_ROOT_ACCESS = 0x20,
  ALLOW_STREAM_OUTPUT = 0x40

CX struct D3D12_ROOT_SIGNATURE_DESC{
  UINT NumParameters;
  const D3D12_ROOT_PARAMETER* pParameters;
  UINT NumStaticSamplers;
  const D3D12_STATIC_SAMPLER_DESC* pStaticSamplers;
  D3D12_ROOT_SIGNATURE_FLAGS Flags; };

HRESULT WINAPI D3D12SerializeRootSignature(
  const D3D12_ROOT_SIGNATURE_DESC* pRootSignature,
  D3D_ROOT_SIGNATURE_VERSION Version, p19
  ID3DBlob** ppBlob, p20
  ID3DBlob** ppErrorBlob p20 );

HRESULT ID3D12Device::CreateRootSignature(
  UINT nodeMask,
  const void* pBlobWithRootSignature,
  SIZE_T blobLengthInBytes,
  REFIID riid,
  void** ppvRootSignature );
pBlobWithRootSignature: ID3DBlob->GetBufferPointer() p20
ppvRootSignature: a ID3D12RootSignature
```

Versioned root signature creation

```
enum D3D12_DESCRIPTOR_RANGE_FLAGS
D3D12_DESCRIPTOR_RANGE_FLAG_X where X is
  NONE = 0,
  DESCRIPTORS_VOLATILE = 0x1,
  DATA_VOLATILE = 0x2,
  DATA_STATIC_WHILE_SET_AT_EXECUTE = 0x4,
  DATA_STATIC = 0x8

CX struct D3D12_DESCRIPTOR_RANGE1 {
  D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
  UINT NumDescriptors;
  UINT BaseShaderRegister;
  UINT RegisterSpace;
  D3D12_DESCRIPTOR_RANGE_FLAGS Flags;
  UINT OffsetInDescriptorsFromTableStart; };

CX struct D3D12_ROOT_DESCRIPTOR_TABLE1{
  UINT NumDescriptorRanges;
  const D3D12_DESCRIPTOR_RANGE1* pDescriptorRanges; };

enum D3D12_ROOT_DESCRIPTOR_FLAGS
D3D12_ROOT_DESCRIPTOR_FLAG_X where X is
  NONE = 0,
  DATA_VOLATILE = 0x2,
  DATA_STATIC_WHILE_SET_AT_EXECUTE = 0x4,
  DATA_STATIC = 0x8

CX struct D3D12_ROOT_DESCRIPTOR1 {
  UINT ShaderRegister;
  UINT RegisterSpace;
  D3D12_ROOT_DESCRIPTOR_FLAGS Flags; };

CX struct D3D12_ROOT_PARAMETER1 {
  D3D12_ROOT_PARAMETER_TYPE ParameterType;
  union {
    D3D12_ROOT_DESCRIPTOR_TABLE1 DescriptorTable;
    D3D12_ROOT_CONSTANTS Constants;
    D3D12_ROOT_DESCRIPTOR1 Descriptor; };
  D3D12_SHADER_VISIBILITY ShaderVisibility; };

struct D3D12_ROOT_SIGNATURE_DESC1{
    UINT NumParameters;
    const D3D12_ROOT_PARAMETER1* pParameters;
    UINT NumStaticSamplers;
    const D3D12_STATIC_SAMPLER_DESC* pStaticSamplers;
    D3D12_ROOT_SIGNATURE_FLAGS Flags; };

CX struct D3D12_VERSIONED_ROOT_SIGNATURE_DESC {
  D3D_ROOT_SIGNATURE_VERSION Version; p19
  union {
    D3D12_ROOT_SIGNATURE_DESC Desc_1_0;
    D3D12_ROOT_SIGNATURE_DESC1 Desc_1_1; }; };

HRESULT WINAPI D3D12SerializeVersionedRootSignature(
  const D3D12_VERSIONED_ROOT_SIGNATURE_DESC* pRootSignature,
  ID3DBlob** ppBlob, p20
  ID3DBlob** ppErrorBlob) p20;
```

Graphics root arguments set-up

```
void ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant(
  UINT RootParameterIndex,
  UINT SrcData,
  UINT DestOffsetIn32BitValues );

void ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants(
  UINT RootParameterIndex,
  UINT Num32BitValuesToSet,
  const void* pSrcData,
  UINT DestOffsetIn32BitValues );
pSrcData: constants data set

void ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );

void ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );
```

```
void ID3D12GraphicsCommandList::SetGraphicsRootUnorderedAccessView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );

void ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable(
  UINT RootParameterIndex,
  D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor p19 );
```

Compute root arguments set-up

```
void ID3D12GraphicsCommandList::SetComputeRoot32BitConstant(
  UINT RootParameterIndex,
  UINT SrcData,
  UINT DestOffsetIn32BitValues );

void ID3D12GraphicsCommandList::SetComputeRoot32BitConstants(
  UINT RootParameterIndex,
  UINT Num32BitValuesToSet,
  const void* pSrcData,
  UINT DestOffsetIn32BitValues );
```
pSrcData: constants data set

```
void ID3D12GraphicsCommandList::SetComputeRootConstantBufferView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );

void ID3D12GraphicsCommandList::SetComputeRootShaderResourceView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );

void ID3D12GraphicsCommandList::SetComputeRootUnorderedAccessView(
  UINT RootParameterIndex,
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation p19 );

void ID3D12GraphicsCommandList::SetComputeRootDescriptorTable(
  UINT RootParameterIndex,
  D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor p19 );
```

Deserialization

```
HRESULT WINAPI D3D12CreateRootSignatureDeserializer(
  pSrcData,
  SIZE_T SrcDataSizeInBytes,
  REFIID pRootSignatureDeserializerInterface,
  void** ppRootSignatureDeserializer );
```
ppRootSignatureDeserializer: a ID3D12RootSignatureDeserializer

```
D3D12_ROOT_SIGNATURE_DESC
ID3D12RootSignatureDeserializer::GetRootSignatureDesc();

HRESULT WINAPI D3D12CreateVersionedRootSignatureDeserializer(
  LPCVOID pSrcData,
  SIZE_T SrcDataSizeInBytes,
  REFIID pRootSignatureDeserializerInterface,
  void** ppRootSignatureDeserializer);
```
ppRootSignatureDeserializer: a D3D12VersionedRootSignatureDeserializer

```
HRESULT ID3D12VersionedRootSignatureDeserializer::
GetRootSignatureDescAtVersion(
  D3D_ROOT_SIGNATURE_VERSION convertToVersion, p19
  const D3D12_VERSIONED_ROOT_SIGNATURE_DESC** ppDesc);

const D3D12_VERSIONED_ROOT_SIGNATURE_DESC*
ID3D12VersionedRootSignatureDeserializer::
GetUnconvertedRootSignatureDesc();

HRESULT ID3D12VersionedRootSignatureDeserializer::
GetRootSignatureDescAtVersion(
  D3D_ROOT_SIGNATURE_VERSION convertToVersion, p19
  const D3D12_VERSIONED_ROOT_SIGNATURE_DESC** ppDesc);

const D3D12_VERSIONED_ROOT_SIGNATURE_DESC*
ID3D12VersionedRootSignatureDeserializer::
GetUnconvertedRootSignatureDesc();
```

Indirect Drawing

Enables multi draw/dispatch indirect, controlling the command buffer flow, increasing draw calls and potentially lowering driver overhead.

Command signature creation

Specifies the indirect parameters to be used.

```
enum D3D12_INDIRECT_ARGUMENT_TYPE
D3D12_INDIRECT_ARGUMENT_TYPE_X where X is
  DRAW,
  DRAW_INDEXED,
  DISPATCH,
  VERTEX_BUFFER_VIEW,
  INDEX_BUFFER_VIEW,
  CONSTANT,
  CONSTANT_BUFFER_VIEW,
  SHADER_RESOURCE_VIEW,
  UNORDERED_ACCESS_VIEW

struct D3D12_INDIRECT_ARGUMENT_DESC{
  D3D12_INDIRECT_ARGUMENT_TYPE Type;
  union{
    struct VertexBuffer{
      UINT Slot; };
    struct Constant{
      UINT RootParameterIndex;
      UINT DestOffsetIn32BitValues;
      UINT Num32BitValuesToSet; };
    struct ConstantBufferView{
      UINT RootParameterIndex; };
    struct ShaderResourceView{
      UINT RootParameterIndex; };
    struct UnorderedAccessView{
      UINT RootParameterIndex; }; }; };

struct D3D12_COMMAND_SIGNATURE_DESC{
  UINT ByteStride;
  UINT NumArgumentDescs;
  const D3D12_INDIRECT_ARGUMENT_DESC* pArgumentDescs;
  UINT NodeMask; };

HRESULT ID3D12Device::CreateCommandSignature(
  const D3D12_COMMAND_SIGNATURE_DESC* pDesc,
  ID3D12RootSignature* pRootSignature,
  REFIID riid,
  void** ppvCommandSignature );
```
ppvCommandSignature: a ID3D12CommandSignature

Command signature execution

Performs indirect draws/dispatches.

```
void ID3D12GraphicsCommandList::ExecuteIndirect(
  ID3D12CommandSignature* pCommandSignature,
  UINT MaxCommandCount,
  ID3D12Resource* pArgumentBuffer,
  UINT64 ArgumentBufferOffset,
  ID3D12Resource* pCountBuffer,
  UINT64 CountBufferOffset );
```

Helpers

```
struct D3D12_DRAW_ARGUMENTS{
  UINT VertexCountPerInstance;
  UINT InstanceCount;
  UINT StartVertexLocation;
  UINT StartInstanceLocation; };
```
Describes parameters for drawing instances.

```
struct D3D12_DRAW_INDEXED_ARGUMENTS{
  UINT IndexCountPerInstance;
  UINT InstanceCount;
  UINT StartIndexLocation;
  INT BaseVertexLocation;
  UINT StartInstanceLocation; };
```
Describes parameters for drawing indexed instances.

```
struct D3D12_DISPATCH_ARGUMENTS{
  UINT ThreadGroupCountX;
  UINT ThreadGroupCountY;
  UINT ThreadGroupCountZ; };
```
Describes dispatch parameters, for use by the compute shader

Pipeline State Object

Represents the state of all currently set shaders ,as well as certain fixed function states and the geometry primitive topology. A PSO is immutable after creation. PSO inheritance rules are different for direct command lists and bundles.

Pipeline state creation

If no PSO is specified in the call, a default initial state is used.

```
CX struct D3D12_SHADER_BYTECODE{
  const void* pShaderBytecode;
  SIZE_T BytecodeLength; };
pShaderBytecode: ID3DBlob->GetBufferPointer() p20

struct D3D12_SO_DECLARATION_ENTRY{
  UINT Stream;
  LPCSTR SemanticName;
  UINT SemanticIndex;
  BYTE StartComponent;
  BYTE ComponentCount;
  BYTE OutputSlot; };

struct D3D12_STREAM_OUTPUT_DESC{
  const D3D12_SO_DECLARATION_ENTRY* pSODeclaration;
  UINT NumEntries;
  const UINT* pBufferStrides;
  UINT NumStrides;
  UINT RasterizedStream; };

enum D3D12_BLEND
D3D12_BLEND_X where X is
  ZERO,
  ONE,
  SRC_COLOR,
  INV_SRC_COLOR,
  SRC_ALPHA,
  INV_SRC_ALPHA,
  DEST_ALPHA,
  INV_DEST_ALPHA,
  DEST_COLOR,
  INV_DEST_COLOR,
  SRC_ALPHA_SAT,
  BLEND_FACTOR,
  INV_BLEND_FACTOR,
  SRC1_COLOR,
  INV_SRC1_COLOR,
  SRC1_ALPHA,
  INV_SRC1_ALPHA

enum D3D12_BLEND_OP
D3D12_BLEND_OP_X where X is
  ADD,
  SUBTRACT,
  REV_SUBTRACT,
  MIN,
  MAX

enum D3D12_LOGIC_OP
D3D12_LOGIC_OP_X where X is
  CLEAR,
  SET,
  COPY,
  COPY_INVERTED,
  NOOP,
  INVERT,
  AND,
  NAND,
  OR,
  NOR,
  XOR,
  EQUIV,
  AND_REVERSE,
  AND_INVERTED,
  OR_REVERSE,
  OR_INVERTED

enum D3D12_COLOR_WRITE_ENABLE
D3D12_COLOR_WRITE_ENABLE_X where X is
  RED = 1,
  GREEN = 2,
  BLUE = 4,
  ALPHA = 8,
  ALL = (((RED|GREEN)|BLUE)|ALPHA)

struct D3D12_RENDER_TARGET_BLEND_DESC{
  BOOL BlendEnable;
  BOOL LogicOpEnable;
  D3D12_BLEND SrcBlend;
  D3D12_BLEND DestBlend;
  D3D12_BLEND_OP BlendOp;
  D3D12_BLEND SrcBlendAlpha;
  D3D12_BLEND DestBlendAlpha;
  D3D12_BLEND_OP BlendOpAlpha;
  D3D12_LOGIC_OP LogicOp;
  UINT8 RenderTargetWriteMask; };
RenderTargetWriteMask: D3D12_COLOR_WRITE_ENABLE

CX struct D3D12_BLEND_DESC{
  BOOL AlphaToCoverageEnable;
  BOOL IndependentBlendEnable;
  D3D12_RENDER_TARGET_BLEND_DESC RenderTarget[8]; };

enum D3D12_FILL_MODE
D3D12_FILL_MODE_X where X is
  WIREFRAME,
  SOLID

enum D3D12_CULL_MODE
D3D12_CULL_MODE_X where X is
  NONE,
  FRONT,
  BACK

enum D3D12_CONSERVATIVE_RASTERIZATION_MODE
D3D12_CONSERVATIVE_RASTERIZATION_MODE_X where X is
  OFF,
  ON

CX struct D3D12_RASTERIZER_DESC{
  D3D12_FILL_MODE FillMode;
  D3D12_CULL_MODE CullMode;
  BOOL FrontCounterClockwise;
  INT DepthBias;
  FLOAT DepthBiasClamp;
  FLOAT SlopeScaledDepthBias;
  BOOL DepthClipEnable;
  BOOL MultisampleEnable;
  BOOL AntialiasedLineEnable;
  UINT ForcedSampleCount;
  D3D12_CONSERVATIVE_RASTERIZATION_MODE ConservativeRaster;
};

enum D3D12_STENCIL_OP
D3D12_STENCIL_OP_X where X is
  KEEP,
  ZERO,
  REPLACE,
  INCR_SAT,
  DECR_SAT,
  INVERT,
  INCR,
  DECR

struct D3D12_DEPTH_STENCILOP_DESC{
  D3D12_STENCIL_OP StencilFailOp;
  D3D12_STENCIL_OP StencilDepthFailOp;
  D3D12_STENCIL_OP StencilPassOp;
  D3D12_COMPARISON_FUNC StencilFunc; p18 };

enum D3D12_DEPTH_WRITE_MASK
D3D12_DEPTH_WRITE_X where X is
  MASK_ZERO,
  MASK_ALL
```

```
CX struct D3D12_DEPTH_STENCIL_DESC{
  BOOL DepthEnable;
  D3D12_DEPTH_WRITE_MASK DepthWriteMask;
  D3D12_COMPARISON_FUNC DepthFunc; p18
  BOOL StencilEnable;
  UINT8 StencilReadMask;
  UINT8 StencilWriteMask;
  D3D12_DEPTH_STENCILOP_DESC FrontFace;
  D3D12_DEPTH_STENCILOP_DESC BackFace; };

enum D3D12_INPUT_CLASSIFICATION
D3D12_INPUT_CLASSIFICATION_PER_X where X is
  VERTEX_DATA,
  INSTANCE_DATA

struct D3D12_INPUT_ELEMENT_DESC{
  LPCSTR SemanticName;
  UINT SemanticIndex;
  DXGI_FORMAT Format; p19
  UINT InputSlot;
  UINT AlignedByteOffset;
  D3D12_INPUT_CLASSIFICATION InputSlotClass;
  UINT InstanceDataStepRate; };

struct D3D12_INPUT_LAYOUT_DESC{
  const D3D12_INPUT_ELEMENT_DESC* pInputElementDescs;
  UINT NumElements; };

enum D3D12_INDEX_BUFFER_STRIP_CUT_VALUE
D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_X where X is
  DISABLED,
  0xFFFF,
  0xFFFFFFFF

enum D3D12_PRIMITIVE_TOPOLOGY_TYPE
D3D12_PRIMITIVE_TOPOLOGY_TYPE_X where X is
  UNDEFINED,
  POINT,
  LINE,
  TRIANGLE,
  PATCH

struct D3D12_CACHED_PIPELINE_STATE{
  const void* pCachedBlob;
  SIZE_T CachedBlobSizeInBytes; };
pCachedBlob: ID3DBlobl->GetBufferPointer() p20

enum D3D12_PIPELINE_STATE_FLAGS
D3D12_PIPELINE_STATE_FLAG_X where X is
  NONE = 0,
  TOOL_DEBUG = 0x1

struct D3D12_GRAPHICS_PIPELINE_STATE_DESC{
  ID3D12RootSignature* pRootSignature;
  D3D12_SHADER_BYTECODE VS;
  D3D12_SHADER_BYTECODE PS;
  D3D12_SHADER_BYTECODE DS;
  D3D12_SHADER_BYTECODE HS;
  D3D12_SHADER_BYTECODE GS;
  D3D12_STREAM_OUTPUT_DESC StreamOutput;
  D3D12_BLEND_DESC BlendState;
  UINT SampleMask;
  D3D12_RASTERIZER_DESC RasterizerState;
  D3D12_DEPTH_STENCIL_DESC DepthStencilState;
  D3D12_INPUT_LAYOUT_DESC InputLayout;
  D3D12_INDEX_BUFFER_STRIP_CUT_VALUE IBStripCutValue;
  D3D12_PRIMITIVE_TOPOLOGY_TYPE PrimitiveTopologyType;
  UINT NumRenderTargets;
  DXGI_FORMAT RTVFormats[8]; p19
  DXGI_FORMAT DSVFormat; p19
  DXGI_SAMPLE_DESC SampleDesc; p20
  UINT NodeMask;
  D3D12_CACHED_PIPELINE_STATE CachedPSO;
  D3D12_PIPELINE_STATE_FLAGS Flags; };

HRESULT ID3D12Device::CreateGraphicsPipelineState(
  const D3D12_GRAPHICS_PIPELINE_STATE_DESC* pDesc,
  REFIID riid,
  void** ppPipelineState );
ppPipelineState: ID3D12PipelineState

struct D3D12_COMPUTE_PIPELINE_STATE_DESC{
  ID3D12RootSignature* pRootSignature;
  D3D12_SHADER_BYTECODE CS;
  UINT NodeMask;
  D3D12_CACHED_PIPELINE_STATE CachedPSO;
  D3D12_PIPELINE_STATE_FLAGS Flags; };

HRESULT ID3D12Device::CreateComputePipelineState(
  const D3D12_COMPUTE_PIPELINE_STATE_DESC* pDesc,
  REFIID riid,
  void** ppPipelineState );
ppPipelineState: ID3D12PipelineState
```

Pipeline state object caching and libraries.

PSO libraries allow to cache compiled PSOs to disk and avoid costly shader compilation during subsequent application runs.

```
HRESULT ID3D12Device1::CreatePipelineLibrary(
  const void* pLibraryBlob,
  SIZE_T BlobLength,
  REFIID riid,
  void** ppPipelineLibrary);

HRESULT ID3D12PipelineLibrary::StorePipeline(
  LPCWSTR pName,
  ID3D12PipelineState* pPipeline);

SIZE_T ID3D12PipelineLibrary::GetSerializedSize();

HRESULT ID3D12PipelineLibrary::Serialize(
  void* pData,
  SIZE_T DataSizeInBytes);

HRESULT ID3D12PipelineState::GetCachedBlob(
  ID3DBlob** ppBlob p20 );

HRESULT ID3D12PipelineLibrary::LoadGraphicsPipeline(
  LPCWSTR pName,
  const D3D12_GRAPHICS_PIPELINE_STATE_DESC* pDesc,
  REFIID riid,
  void** ppPipelineState);

HRESULT ID3D12PipelineLibrary::LoadComputePipeline(
  LPCWSTR pName,
  const D3D12_COMPUTE_PIPELINE_STATE_DESC* pDesc,
  REFIID riid,
  void** ppPipelineState);
```

Resources

Encapsulates a generalized ability of the CPU and GPU to read and write to physical memory, or heaps. It contains abstractions for organizing and manipulating simple arrays of data as well as multidimensional data optimized for shader sampling.

Descriptor heaps

A descriptor heap is a collection of contiguous allocations of descriptors.

```
enum D3D12_DESCRIPTOR_HEAP_TYPE
D3D12_DESCRIPTOR_HEAP_TYPE_X where X is
  CBV_SRV_UAV,
  SAMPLER,
  RTV,
  DSV,
  NUM_TYPES

enum D3D12_DESCRIPTOR_HEAP_FLAGS
D3D12_DESCRIPTOR_HEAP_FLAG_X where X is
  NONE = 0,
  SHADER_VISIBLE = 0x1

struct D3D12_DESCRIPTOR_HEAP_DESC{
  D3D12_DESCRIPTOR_HEAP_TYPE Type;
  UINT NumDescriptors;
  D3D12_DESCRIPTOR_HEAP_FLAGS Flags;
  UINT NodeMask; };
```

```
D3D12_DESCRIPTOR_HEAP_DESC ID3D12DescriptorHeap::GetDesc();

HRESULT ID3D12Device::CreateDescriptorHeap(
  const D3D12_DESCRIPTOR_HEAP_DESC* pDescriptorHeapDesc,
  REFIID riid,
  void** ppvHeap );
ppvHeap: ID3D12DescriptorHeap
```

Descriptor copy

```
void ID3D12Device::CopyDescriptors(
  UINT NumDestDescriptorRanges,
  const D3D12_CPU_DESCRIPTOR_HANDLE* pDestDescriptorRangeStarts, p18
  const UINT* pDestDescriptorRangeSizes,
  UINT NumSrcDescriptorRanges,
  const D3D12_CPU_DESCRIPTOR_HANDLE* pSrcDescriptorRangeStarts, p18
  const UINT* pSrcDescriptorRangeSizes,
  D3D12_DESCRIPTOR_HEAP_TYPE  DescriptorHeapsType );

void ID3D12Device::CopyDescriptorsSimple(
  UINT NumDescriptors,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptorRangeStart, p18
  D3D12_CPU_DESCRIPTOR_HANDLE SrcDescriptorRangeStart, p18
  D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapsType );
```

Views creation

Views are built to describe resources capabilities and dimensions.

```
enum D3D12_DSV_DIMENSION
D3D12_DSV_DIMENSION_X where X is
  UNKNOWN,
  TEXTURE1D,
  TEXTURE1DARRAY,
  TEXTURE2D,
  TEXTURE2DARRAY,
  TEXTURE2DMS,
  TEXTURE2DMSARRAY

struct D3D12_TEX1D_DSV{
  UINT MipSlice; };

struct D3D12_TEX1D_ARRAY_DSV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX2D_DSV{
  UINT MipSlice; };

struct D3D12_TEX2D_ARRAY_DSV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX2DMS_DSV{
  UINT UnusedField_NothingToDefine; };

struct D3D12_TEX2DMS_ARRAY_DSV{
  UINT FirstArraySlice;
  UINT ArraySize; };

enum D3D12_DSV_FLAGS
D3D12_DSV_FLAG_X where X is
  NONE = 0,
  READ_ONLY_DEPTH = 0x1,
  READ_ONLY_STENCIL = 0x2

struct D3D12_DEPTH_STENCIL_VIEW_DESC{
  DXGI_FORMAT Format; p19
  D3D12_DSV_DIMENSION ViewDimension;
  D3D12_DSV_FLAGS Flags;
  union{
    D3D12_TEX1D_DSV Texture1D;
    D3D12_TEX1D_ARRAY_DSV Texture1DArray;
    D3D12_TEX2D_DSV Texture2D;
    D3D12_TEX2D_ARRAY_DSV Texture2DArray;
    D3D12_TEX2DMS_DSV Texture2DMS;
    D3D12_TEX2DMS_ARRAY_DSV Texture2DMSArray; }; };

void ID3D12Device::CreateDepthStencilView(
  ID3D12Resource* pResource,
  const D3D12_DEPTH_STENCIL_VIEW_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );

enum D3D12_RTV_DIMENSION
D3D12_RTV_DIMENSION_X where X is
  UNKNOWN,
  BUFFER,
  TEXTURE1D,
  TEXTURE1DARRAY,
  TEXTURE2D,
  TEXTURE2DARRAY,
  TEXTURE2DMS,
  TEXTURE2DMSARRAY,
  TEXTURE3D

struct D3D12_BUFFER_RTV{
  UINT64 FirstElement;
  UINT NumElements; };

struct D3D12_TEX1D_RTV{
  UINT MipSlice; };

struct D3D12_TEX1D_ARRAY_RTV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX2D_RTV{
  UINT MipSlice;
  UINT PlaneSlice; };

struct D3D12_TEX2D_ARRAY_RTV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize;
  UINT PlaneSlice; };

struct D3D12_TEX2DMS_RTV{
  UINT UnusedField_NothingToDefine; };

struct D3D12_TEX2DMS_ARRAY_RTV{
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX3D_RTV{
  UINT MipSlice;
  UINT FirstWSlice;
  UINT WSize; };

struct D3D12_RENDER_TARGET_VIEW_DESC{
  DXGI_FORMAT Format; p19
  D3D12_RTV_DIMENSION ViewDimension;
  union{
    D3D12_BUFFER_RTV Buffer;
    D3D12_TEX1D_RTV Texture1D;
    D3D12_TEX1D_ARRAY_RTV Texture1DArray;
    D3D12_TEX2D_RTV Texture2D;
    D3D12_TEX2D_ARRAY_RTV Texture2DArray;
    D3D12_TEX2DMS_RTV Texture2DMS;
    D3D12_TEX2DMS_ARRAY_RTV Texture2DMSArray;
    D3D12_TEX3D_RTV Texture3D; }; };

void ID3D12Device::CreateRenderTargetView(
  ID3D12Resource* pResource,
  const D3D12_RENDER_TARGET_VIEW_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );

struct D3D12_CONSTANT_BUFFER_VIEW_DESC{
  D3D12_GPU_VIRTUAL_ADDRESS BufferLocation; p19
  UINT SizeInBytes; };

void ID3D12Device::CreateConstantBufferView(
  const D3D12_CONSTANT_BUFFER_VIEW_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );
```

```
enum D3D12_SRV_DIMENSION
D3D12_SRV_DIMENSION_X where X is
  UNKNOWN,
  BUFFER,
  TEXTURE1D,
  TEXTURE1DARRAY,
  TEXTURE2D,
  TEXTURE2DARRAY,
  TEXTURE2DMS,
  TEXTURE2DMSARRAY,
  TEXTURE3D,
  TEXTURECUBE,
  TEXTURECUBEARRAY

enum D3D12_SHADER_COMPONENT_MAPPING
D3D12_SHADER_COMPONENT_MAPPING_X where X is
  FROM_MEMORY_COMPONENT_0,
  FROM_MEMORY_COMPONENT_1,
  FROM_MEMORY_COMPONENT_2,
  FROM_MEMORY_COMPONENT_3,
  FORCE_VALUE_0,
  FORCE_VALUE_1
The default 1:1 mapping can be indicated by specifying
D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING, otherwise an
arbitrary mapping can be specified using the macro
D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING

enum D3D12_BUFFER_SRV_FLAGS
D3D12_BUFFER_SRV_FLAG_X where X is
  NONE = 0,
  RAW = 0x1

struct D3D12_BUFFER_SRV{
  UINT64 FirstElement;
  UINT NumElements;
  UINT StructureByteStride;
  D3D12_BUFFER_SRV_FLAGS Flags; };

struct D3D12_TEX1D_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEX1D_ARRAY_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  UINT FirstArraySlice;
  UINT ArraySize;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEX2D_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  UINT PlaneSlice;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEX2D_ARRAY_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  UINT FirstArraySlice;
  UINT ArraySize;
  UINT PlaneSlice;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEX2DMS_SRV{
  UINT UnusedField_NothingToDefine; };

struct D3D12_TEX2DMS_ARRAY_SRV{
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX3D_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEXCUBE_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  FLOAT ResourceMinLODClamp; };

struct D3D12_TEXCUBE_ARRAY_SRV{
  UINT MostDetailedMip;
  UINT MipLevels;
  UINT First2DArrayFace;
  UINT NumCubes;
  FLOAT ResourceMinLODClamp; };

struct D3D12_SHADER_RESOURCE_VIEW_DESC{
  DXGI_FORMAT Format; p19
  D3D12_SRV_DIMENSION ViewDimension;
  UINT Shader4ComponentMapping;
  union{
    D3D12_BUFFER_SRV Buffer;
    D3D12_TEX1D_SRV Texture1D;
    D3D12_TEX1D_ARRAY_SRV Texture1DArray;
    D3D12_TEX2D_SRV Texture2D;
    D3D12_TEX2D_ARRAY_SRV Texture2DArray;
    D3D12_TEX2DMS_SRV Texture2DMS;
    D3D12_TEX2DMS_ARRAY_SRV Texture2DMSArray;
    D3D12_TEX3D_SRV Texture3D;
    D3D12_TEXCUBE_SRV TextureCube;
    D3D12_TEXCUBE_ARRAY_SRV TextureCubeArray; }; };
Shader4ComponentMapping: D3D12_SHADER_COMPONENT_MAPPING

void ID3D12Device::CreateShaderResourceView(
  ID3D12Resource* pResource,
  const D3D12_SHADER_RESOURCE_VIEW_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );

enum D3D12_UAV_DIMENSION
D3D12_UAV_DIMENSION_X where X is
  UNKNOWN,
  BUFFER,
  TEXTURE1D,
  TEXTURE1DARRAY,
  TEXTURE2D,
  TEXTURE2DARRAY,
  TEXTURE3D

enum D3D12_BUFFER_UAV_FLAGS
D3D12_BUFFER_UAV_FLAG_X where X is
  NONE = 0,
  RAW = 0x1

struct D3D12_BUFFER_UAV{
  UINT64 FirstElement;
  UINT NumElements;
  UINT StructureByteStride;
  UINT64 CounterOffsetInBytes;
  D3D12_BUFFER_UAV_FLAGS Flags; };

struct D3D12_TEX1D_UAV{
  UINT MipSlice; };

struct D3D12_TEX1D_ARRAY_UAV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize; };

struct D3D12_TEX2D_UAV{
  UINT MipSlice;
  UINT PlaneSlice; };

struct D3D12_TEX2D_ARRAY_UAV{
  UINT MipSlice;
  UINT FirstArraySlice;
  UINT ArraySize;
  UINT PlaneSlice; };

struct D3D12_TEX3D_UAV{
  UINT MipSlice;
  UINT FirstWSlice;
  UINT WSize; };
```

```
struct D3D12_UNORDERED_ACCESS_VIEW_DESC{
  DXGI_FORMAT Format; p19
  D3D12_UAV_DIMENSION ViewDimension;
  union{
    D3D12_BUFFER_UAV Buffer;
    D3D12_TEX1D_UAV Texture1D;
    D3D12_TEX1D_ARRAY_UAV Texture1DArray;
    D3D12_TEX2D_UAV Texture2D;
    D3D12_TEX2D_ARRAY_UAV Texture2DArray;
    D3D12_TEX3D_UAV Texture3D; }; };

void ID3D12Device::CreateUnorderedAccessView(
  ID3D12Resource* pResource,
  ID3D12Resource* pCounterResource,
  const D3D12_UNORDERED_ACCESS_VIEW_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );

struct D3D12_SAMPLER_DESC{
  D3D12_FILTER Filter; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressU; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressV; p19
  D3D12_TEXTURE_ADDRESS_MODE AddressW; p19
  FLOAT MipLODBias;
  UINT MaxAnisotropy;
  D3D12_COMPARISON_FUNC ComparisonFunc; p18
  FLOAT BorderColor[4];
  FLOAT MinLOD;
  FLOAT MaxLOD; };

void ID3D12Device::CreateSampler(
  const D3D12_SAMPLER_DESC* pDesc,
  D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor p18 );
```

Resource copy and subresources

Resources are divided into one ore multiple resources. Some API access an entire resource while other only a portion of it.

```
CX struct D3D12_RANGE{
  SIZE_T Begin;
  SIZE_T End; };

struct D3D12_MEMCPY_DEST{
  void* pData;
  SIZE_T RowPitch;
  SIZE_T SlicePitch; };
Helper, describes the destination of a memory copy operation

HRESULT ID3D12Resource::Map(
  UINT Subresource,
  const D3D12_RANGE* pReadRange,
  void** ppData );
ppData: resource data.

void ID3D12Resource::Unmap(
  UINT Subresource,
  const D3D12_RANGE* pWrittenRange );

void ID3D12GraphicsCommandList::CopyResource(
  ID3D12Resource* pDstResource,
  ID3D12Resource* pSrcResource );

void ID3D12GraphicsCommandList::CopyBufferRegion(
  ID3D12Resource* pDstBuffer,
  UINT64 DstOffset,
  ID3D12Resource* pSrcBuffer,
  UINT64 SrcOffset,
  UINT64 NumBytes );

CX struct D3D12_SUBRESOURCE_FOOTPRINT{
  DXGI_FORMAT Format; p19
  UINT Width;
  UINT Height;
  UINT Depth;
  UINT RowPitch; };

struct D3D12_PLACED_SUBRESOURCE_FOOTPRINT{
  UINT64 Offset;
  D3D12_SUBRESOURCE_FOOTPRINT Footprint; };

void ID3D12Device::GetCopyableFootprints(
  const D3D12_RESOURCE_DESC* pResourceDesc, p19
  UINT FirstSubresource,
  UINT NumSubresources,
  UINT64 BaseOffset,
  D3D12_PLACED_SUBRESOURCE_FOOTPRINT* pLayouts,
  UINT* pNumRows,
  UINT64* pRowSizeInBytes,
  UINT64* pTotalBytes );

struct D3D12_SUBRESOURCE_INFO{
  UINT64 Offset;
  UINT RowPitch;
  UINT DepthPitch; };
Helper. describes subresource data.

struct D3D12_SUBRESOURCE_DATA{
  const void* pData;
  LONG_PTR RowPitch;
  LONG_PTR SlicePitch; };
Helper, describes subresource data.

enum D3D12_TEXTURE_COPY_TYPE
D3D12_TEXTURE_COPY_TYPE_X where X is
  SUBRESOURCE_INDEX,
  PLACED_FOOTPRINT

CX struct D3D12_TEXTURE_COPY_LOCATION{
  ID3D12Resource* pResource;
  D3D12_TEXTURE_COPY_TYPE Type;
  union{
    D3D12_PLACED_SUBRESOURCE_FOOTPRINT PlacedFootprint;
    UINT SubresourceIndex; }; };

CX struct D3D12_BOX{
  UINT left;
  UINT top;
  UINT front;
  UINT right;
  UINT bottom;
  UINT back; };

void ID3D12GraphicsCommandList::CopyTextureRegion(
  const D3D12_TEXTURE_COPY_LOCATION* pDst,
  UINT DstX,
  UINT DstY,
  UINT DstZ,
  const D3D12_TEXTURE_COPY_LOCATION* pSrc,
  const D3D12_BOX* pSrcBox );

void ID3D12GraphicsCommandList::ResolveSubresource(
  ID3D12Resource* pDstResource,
  UINT DstSubresource,
  ID3D12Resource* pSrcResource,
  UINT SrcSubresource,
  DXGI_FORMAT Format p19 );

HRESULT ID3D12Resource::WriteToSubresource(
  UINT DstSubresource,
  const D3D12_BOX* pDstBox,
  const void* pSrcData,
  UINT SrcRowPitch,
  UINT SrcDepthPitch );
pSrcData: source data in memory.

HRESULT ID3D12Resource::ReadFromSubresource(
  void* pDstData,
  UINT DstRowPitch,
  UINT DstDepthPitch,
  UINT SrcSubresource,
  const D3D12_BOX* pSrcBox );
pDstData: destination data in memory.
```

Allocation

Heap Creation

A heap is an abstraction of contiguous memory allocation, used to manage physical memory.

```
enum D3D12_HEAP_TYPE
D3D12_HEAP_TYPE_X where X is
  DEFAULT,
  UPLOAD,
  READBACK,
  CUSTOM

enum D3D12_CPU_PAGE_PROPERTY
D3D12_CPU_PAGE_PROPERTY_X where X is
  UNKNOWN,
  NOT_AVAILABLE,
  WRITE_COMBINE2,
  WRITE_BACK

enum D3D12_MEMORY_POOL
D3D12_MEMORY_POOL_X where X is
  UNKNOWN,
  L0,
  L1

CX struct D3D12_HEAP_PROPERTIES{
  D3D12_HEAP_TYPE Type;
  D3D12_CPU_PAGE_PROPERTY CPUPageProperty;
  D3D12_MEMORY_POOL MemoryPoolPreference;
  UINT CreationNodeMask;
  UINT VisibleNodeMask; };

enum D3D12_HEAP_FLAGS
D3D12_HEAP_FLAG_X where X is
  NONE = 0,
  SHARED = 0x1,
  DENY_BUFFERS = 0x4,
  ALLOW_DISPLAY = 0x8,
  SHARED_CROSS_ADAPTER = 0x20,
  DENY_RT_DS_TEXTURES = 0x40,
  DENY_NON_RT_DS_TEXTURES = 0x80,
  ALLOW_ALL_BUFFERS_AND_TEXTURES = 0,
  ALLOW_ONLY_BUFFERS = 0xc0,
  ALLOW_ONLY_NON_RT_DS_TEXTURES = 0x44,
  ALLOW_ONLY_RT_DS_TEXTURES = 0x84
  HARDWARE_PROTECTED = 0x100,

CX struct D3D12_HEAP_DESC{
  UINT64 SizeInBytes;
  D3D12_HEAP_PROPERTIES Properties;
  UINT64 Alignment;
  D3D12_HEAP_FLAGS Flags; };

HRESULT ID3D12Device::CreateHeap(
  const D3D12_HEAP_DESC* pDesc,
  REFIID riid,
  void** ppvHeap );
ppvHeap: the heap

D3D12_HEAP_DESC ID3D12Heap::GetDesc();

HRESULT ID3D12Resource::GetHeapProperties(
  D3D12_HEAP_PROPERTIES* pHeapProperties,
  D3D12_HEAP_FLAGS* pHeapFlags );

D3D12_HEAP_PROPERTIES ID3D12Device::GetCustomHeapProperties(
  UINT nodeMask,
  D3D12_HEAP_TYPE heapType );
```

Resource allocation

Committed resources are the most common idea of D3D resources over the generations: they create both a resource and an implicit appropriately sized heap at the same time. The implicit heap properties must be passed to match functional parity with previous D3D versions.

```
CX struct D3D12_RESOURCE_ALLOCATION_INFO{
  UINT64 SizeInBytes;
  UINT64 Alignment; };

D3D12_RESOURCE_ALLOCATION_INFO
ID3D12Device::GetResourceAllocationInfo(
  UINT visibleMask,
  UINT numResourceDescs,
  const D3D12_RESOURCE_DESC* pResourceDescs p19 );

HRESULT ID3D12Device::CreateCommittedResource(
  const D3D12_HEAP_PROPERTIES* pHeapProperties,
  D3D12_HEAP_FLAGS HeapFlags,
  const D3D12_RESOURCE_DESC* pResourceDesc, p19
  D3D12_RESOURCE_STATES InitialResourceState, p19
  const D3D12_CLEAR_VALUE* pOptimizedClearValue, p19
  REFIID riidResource,
  void** ppvResource );
ppvResource: resorce address in a memory block
```

Resource sub-allocation

Placed resources allow the placement of a resource at a non-zero offset within a heap. Multiple resources may overlap, and the application must use the TiledResourceBarrier to re-use physical memory correctly

```
HRESULT ID3D12Device::CreatePlacedResource(
  ID3D12Heap* pHeap,
  UINT64 HeapOffset,
  const D3D12_RESOURCE_DESC* pDesc, p19
  D3D12_RESOURCE_STATES InitialState, p19
  const D3D12_CLEAR_VALUE* pOptimizedClearValue, p18
  REFIID riid,
  void** ppvResource );
ppvResource: resorce address in a memory block
```

Tiling

Allowed by reserved resources (also known as tiled resources) having their own unique GPU virtual address space. The VA space is contiguous and it can be sparsely mapped to and reconfigured on the fly in different times. After creation, a reserved resource is not yet mapped to any pages in a heap, but it must be mapped to physical memory using CopyTileMappings and UpdateTileMappings.

```
HRESULT ID3D12Device::CreateReservedResource(
  const D3D12_RESOURCE_DESC* pDesc, p19
  D3D12_RESOURCE_STATES InitialState, p19
  const D3D12_CLEAR_VALUE* pOptimizedClearValue, p18
  REFIID riid,
  void** ppvResource );
ppvResource: retrieved resource

CX struct D3D12_TILED_RESOURCE_COORDINATE{
  UINT X;
  UINT Y;
  UINT Z;
  UINT Subresource; };

CX struct D3D12_TILE_REGION_SIZE{
  UINT NumTiles;
  BOOL UseBox;
  UINT Width;
  UINT16 Height;
  UINT16 Depth; };

enum D3D12_TILE_RANGE_FLAGS
X where X is
  NONE,
  NULL,
  SKIP,
  REUSE_SINGLE_TILE
```

```
enum D3D12_TILE_MAPPING_FLAGS
D3D12_TILE_MAPPING_FLAG_X where X is
  NONE = 0,
  NO_HAZARD = 0x1

void ID3D12CommandQueue::UpdateTileMappings(
  ID3D12Resource* pResource,
  UINT NumResourceRegions,
  const D3D12_TILED_RESOURCE_COORDINATE* pResourceRegionStartCoordinates,
  const D3D12_TILE_REGION_SIZE* pResourceRegionSizes,
  ID3D12Heap* pHeap,
  UINT NumRanges,
  const D3D12_TILE_RANGE_FLAGS* pRangeFlags,
  const UINT* pHeapRangeStartOffsets,
  const UINT* pRangeTileCounts,
  D3D12_TILE_MAPPING_FLAGS Flags );

void ID3D12CommandQueue::CopyTileMappings(
  ID3D12Resource* pDstResource,
  const D3D12_TILED_RESOURCE_COORDINATE* pDstRegionStartCoordinate,
  ID3D12Resource* pSrcResource,
  const D3D12_TILED_RESOURCE_COORDINATE* pSrcRegionStartCoordinate,
  const D3D12_TILE_REGION_SIZE* pRegionSize,
  D3D12_TILE_MAPPING_FLAGS Flags )

CX struct D3D12_PACKED_MIP_INFO{
  UINT8 NumStandardMips;
  UINT8 NumPackedMips;
  UINT NumTilesForPackedMips;
  UINT StartTileIndexInOverallResource; };

CX struct D3D12_TILE_SHAPE{
  UINT WidthInTexels;
  UINT HeightInTexels;
  UINT DepthInTexels; };

CX struct D3D12_SUBRESOURCE_TILING{
  UINT WidthInTiles;
  UINT16 HeightInTiles;
  UINT16 DepthInTiles;
  UINT StartTileIndexInOverallResource; };

void ID3D12Device::GetResourceTiling(
  ID3D12Resource* pTiledResource,
  UINT* pNumTilesForEntireResource,
  D3D12_PACKED_MIP_INFO* pPackedMipDesc,
  D3D12_TILE_SHAPE* pStandardTileShapeForNonPackedMips,
  UINT* pNumSubresourceTilings,
  UINT FirstSubresourceTilingToGet,
  D3D12_SUBRESOURCE_TILING* pSubresourceTilingsForNonPackedMips );

enum D3D12_TILE_COPY_FLAGS
D3D12_TILE_COPY_FLAG_X where X is
  NONE = 0,
  NO_HAZARD = 0x1,
  LINEAR_BUFFER_TO_SWIZZLED_TILED_RESOURCE = 0x2,
  SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER = 0x4

void ID3D12GraphicsCommandList::CopyTiles(
  ID3D12Resource* pTiledResource,
  const D3D12_TILED_RESOURCE_COORDINATE* pTileRegionStartCoordinate,
  const D3D12_TILE_REGION_SIZE* pTileRegionSize,
  ID3D12Resource* pBuffer,
  UINT64 BufferStartOffsetInBytes,
  D3D12_TILE_COPY_FLAGS Flags );
```

Shared Handles

Sharing heap, resources and fences is needed for proper multi-adapter application architecture handling.

```
HRESULT ID3D12Device::CreateSharedHandle(
  ID3D12DeviceChild* pObject,
  const SECURITY_ATTRIBUTES* pAttributes,
  DWORD Access,
  LPCWSTR Name,
  HANDLE* pHandle );
Access: GENERIC_ALL

HRESULT ID3D12Device::OpenSharedHandle(
  HANDLE NTHandle,
  REFIID riid,
  void** ppvObj );
ppvObj: ID3D12Heap, ID3D12Resource, ID3D12Fence

HRESULT ID3D12Device::OpenSharedHandleByName(
  LPCWSTR Name,
  DWORD Access,
  HANDLE* pNTHandle );
Access: GENERIC_ALL
```

Memory Reservation

Allows to inform the OS of the amount of physical video memory the application cannot go without. The amount of physical memory available can fluctuate noticeably due user and OS activities. A resource is resident when it is accessible by the GPU.

Memory reservation capabilities

```
enum DXGI_MEMORY_SEGMENT_GROUP
DXGI_MEMORY_SEGMENT_GROUP_X where X is
  LOCAL,
  NON_LOCAL

struct DXGI_QUERY_VIDEO_MEMORY_INFO{
  UINT64 Budget;
  UINT64 CurrentUsage;
  UINT64 AvailableForReservation;
  UINT64 CurrentReservation; };

HRESULT IDXGIAdapter3::QueryVideoMemoryInfo(
  UINT NodeIndex,
  DXGI_MEMORY_SEGMENT_GROUP MemorySegmentGroup,
  DXGI_QUERY_VIDEO_MEMORY_INFO* pVideoMemoryInfo );

HRESULT IDXGIAdapter3::RegisterVideoMemoryBudgetChangeNotificationEvent(
  HANDLE hEvent,
  DWORD* pdwCookie );

void IDXGIAdapter3::UnregisterVideoMemoryBudgetChangeNotification(
  DWORD dwCookie );
```

Reservation and residency management

```
HRESULT IDXGIAdapter3::SetVideoMemoryReservation(
  UINT NodeIndex,
  DXGI_MEMORY_SEGMENT_GROUP MemorySegmentGroup,
  UINT64 Reservation );

HRESULT ID3D12Device::MakeResident(
  UINT NumObjects,
  ID3D12Pageable* const* ppObjects );

HRESULT ID3D12Device::Evict(
  UINT NumObjects,
  ID3D12Pageable* const* ppObjects );
```

Reserved for future updates

This method is reserved for future use, currently it always returns a failure and causes device-removal.

```
enum D3D12_RESIDENCY_PRIORITY
D3D12_RESIDENCY_PRIORITY_X where X is
  MINIMUM,
  LOW,
  NORMAL,
  HIGH,
  MAXIMUM

HRESULT ID3D12Device1::SetResidencyPriority(
  UINT NumObjects,
  ID3D12Pageable* const* ppObjects,
  const D3D12_RESIDENCY_PRIORITY* pPriorities);
```

Queries

Are grouped into arrays of queries called a query heap. A query heap has a type which defines the valid types of queries that can be used with that heap.

```
enum D3D12_QUERY_HEAP_TYPE
D3D12_QUERY_HEAP_TYPE_X where X is
  OCCLUSION,
  TIMESTAMP,
  PIPELINE_STATISTICS,
  SO_STATISTICS
Where value is mapped as follows:
```

OCCLUSION	0 or 1
TIMESTAMP	high performance timing data
PIPELINE_STATISTICS	D3D12_QUERY_DATA_PIPELINE_STATISTICS
SO_STATISTICS	D3D12_QUERY_DATA_SO_STATISTICS

```
struct D3D12_QUERY_DATA_PIPELINE_STATISTICS{
  UINT64 IAVertices;
  UINT64 IAPrimitives;
  UINT64 VSInvocations;
  UINT64 GSInvocations;
  UINT64 GSPrimitives;
  UINT64 CInvocations;
  UINT64 CPrimitives;
  UINT64 PSInvocations;
  UINT64 HSInvocations;
  UINT64 DSInvocations;
  UINT64 CSInvocations; };

struct D3D12_QUERY_DATA_SO_STATISTICS{
  UINT64 NumPrimitivesWritten;
  UINT64 PrimitivesStorageNeeded; };

struct D3D12_QUERY_HEAP_DESC{
  D3D12_QUERY_HEAP_TYPE Type;
  UINT Count;
  UINT NodeMask; };

HRESULT ID3D12Device::CreateQueryHeap(
  const D3D12_QUERY_HEAP_DESC* pDesc,
  REFIID riid,
  void** ppvHeap );
ppvHeap: the created heap

void ID3D12GraphicsCommandList::BeginQuery(
  ID3D12QueryHeap* pQueryHeap,
  D3D12_QUERY_TYPE Type,
  UINT Index );

void ID3D12GraphicsCommandList::EndQuery(
  ID3D12QueryHeap* pQueryHeap,
  D3D12_QUERY_TYPE Type,
  UINT Index );

void ID3D12GraphicsCommandList::ResolveQueryData(
  ID3D12QueryHeap* pQueryHeap,
  D3D12_QUERY_TYPE Type,
  UINT StartIndex,
  UINT NumQueries,
  ID3D12Resource* pDestinationBuffer,
  UINT64 AlignedDestinationBufferOffset );
```

Timing

Applications can query the GPU timestamp frequency on a per-command queue basis. The returned frequency is measured in Hz (ticks/sec). Note that GPU (dedicated or integrated) timestamp counters are not necessarily directly related to the clock speed of these processors, but instead work from timestamp ticks.

Adapter timing

```
HRESULT ID3D12CommandQueue::GetTimestampFrequency(
  UINT64* pFrequency );

HRESULT ID3D12CommandQueue::GetClockCalibration(
  UINT64* pGpuTimestamp,
  UINT64* pCpuTimestamp );

HRESULT ID3D12Device::SetStablePowerState(
  BOOL Enable );
Do not call this method in normal execution for a shipped
application. This method only works while the machine is in
developer mode.
```

Output and presentation timing

```
struct DXGI_FRAME_STATISTICS{
  UINT PresentCount;
  UINT PresentRefreshCount;
  UINT SyncRefreshCount;
  LARGE_INTEGER SyncQPCTime;
  LARGE_INTEGER SyncGPUTime; };

HRESULT IDXGIOutput::GetFrameStatistics(
  DXGI_FRAME_STATISTICS* pStats );
Works only in full-screen mode, prefer
IDXGISwapChain::GetFrameStatistics instead.

HRESULT IDXGISwapChain::GetFrameStatistics(
  DXGI_FRAME_STATISTICS* pStats );

HRESULT IDXGISwapChain::GetLastPresentCount(
  UINT* pLastPresentCount );

HRESULT IDXGIOutput::WaitForVBlank();
Used on Valve OpenVR to measure output VSync time gap.
```

Presentation

Presentation happens manuallly, controlling the back-buffers rotation with the swap-chain.

Window association

```
HRESULT IDXGIFactory::MakeWindowAssociation(
  HWND WindowHandle,
  UINT Flags );
Flags: a combination of the following values
  DXGI_MWA_NO_WINDOW_CHANGES,
  DXGI_MWA_NO_ALT_ENTER,
  DXGI_MWA_NO_PRINT_SCREEN

HRESULT IDXGIFactory::GetWindowAssociation(
  HWND* pWindowHandle );
```

Swap-Chain creation

```
enum DXGI_FEATURE
DXGI_FEATURE_X where X is
  PRESENT_ALLOW_TEARING = 0

HRESULT IDXGIFactory5::CheckFeatureSupport(
  DXGI_FEATURE Feature,
  void* pFeatureSupportData,
  UINT FeatureSupportDataSize );
pFeatureSupportData: a BOOL flag

DXGI_USAGE constants
DXGI_X where X is
  CPU_ACCESS_NONE = 0
  CPU_ACCESS_DYNAMIC = 1
  CPU_ACCESS_READ_WRITE = 2
  CPU_ACCESS_SCRATCH = 3
  CPU_ACCESS_FIELD = 15
  USAGE_SHADER_INPUT = 0x00000010UL
  USAGE_RENDER_TARGET_OUTPUT = 0x00000020UL
  USAGE_BACK_BUFFER = 0x00000040UL
  USAGE_SHARED = 0x00000080UL
  USAGE_READ_ONLY = 0x00000100UL
  USAGE_DISCARD_ON_PRESENT = 0x00000200UL
  USAGE_UNORDERED_ACCESS = 0x00000400UL
```

```
enum DXGI_SWAP_EFFECT
DXGI_SWAP_EFFECT_X where X is
  DISCARD, N.A.
  SEQUENTIAL, N.A.
  FLIP_SEQUENTIAL,
  FLIP_DISCARD

enum DXGI_SWAP_CHAIN_FLAG
DXGI_SWAP_CHAIN_FLAG_X where X is
  NONPREROTATED = 1,
  ALLOW_MODE_SWITCH = 2,
  GDI_COMPATIBLE = 4,
  RESTRICTED_CONTENT = 8,
  RESTRICT_SHARED_RESOURCE_DRIVER = 16,
  DISPLAY_ONLY = 32,
  FRAME_LATENCY_WAITABLE_OBJECT = 64,
  FOREGROUND_LAYER = 128,
  FULLSCREEN_VIDEO = 256,
  YUV_VIDEO = 512,
  HW_PROTECTED = 1024
  ALLOW_TEARING = 2048

struct DXGI_SWAP_CHAIN_DESC{
  DXGI_MODE_DESC BufferDesc; p20
  DXGI_SAMPLE_DESC SampleDesc; p20
  DXGI_USAGE BufferUsage;
  UINT BufferCount;
  HWND OutputWindow;
  BOOL Windowed;
  DXGI_SWAP_EFFECT SwapEffect;
  UINT Flags; };
Flags: a combination of DXGI_SWAP_CHAIN_FLAG values

HRESULT IDXGIFactory::CreateSwapChain(
  IUnknown* pDevice,
  DXGI_SWAP_CHAIN_DESC* pDesc,
  IDXGISwapChain** ppSwapChain );

enum DXGI_SCALING
DXGI_SCALING_X where X is
  STRETCH,
  NONE,
  ASPECT_RATIO_STRETCH

enum DXGI_ALPHA_MODE
DXGI_ALPHA_MODE_X where X is
  UNSPECIFIED,
  PREMULTIPLIED,
  STRAIGHT,
  IGNORE,
  FORCE_DWORD N.A.

struct DXGI_SWAP_CHAIN_DESC1{
  UINT Width;
  UINT Height;
  DXGI_FORMAT Format; p19
  BOOL Stereo;
  DXGI_SAMPLE_DESC SampleDesc; p20
  DXGI_USAGE BufferUsage;
  UINT BufferCount;
  DXGI_SCALING Scaling;
  DXGI_SWAP_EFFECT SwapEffect;
  DXGI_ALPHA_MODE AlphaMode;
  UINT Flags; };
Flags: a combination of DXGI_SWAP_CHAIN_FLAG values

struct DXGI_SWAP_CHAIN_FULLSCREEN_DESC{
  DXGI_RATIONAL RefreshRate; p20
  DXGI_MODE_SCANLINE_ORDER ScanlineOrdering; p20
  DXGI_MODE_SCALING Scaling; p20
  BOOL Windowed; };

HRESULT IDXGIFactory2::CreateSwapChainForHwnd(
  IUnknown* pDevice,
  HWND hWnd,
  const DXGI_SWAP_CHAIN_DESC1* pDesc,
  const DXGI_SWAP_CHAIN_FULLSCREEN_DESC* pFullscreenDesc,
  IDXGIOutput* pRestrictToOutput,
  IDXGISwapChain1** ppSwapChain );

HRESULT IDXGIFactory2::CreateSwapChainForCoreWindow(
  IUnknown* pDevice,
  IUnknown* pWindow,
  const DXGI_SWAP_CHAIN_DESC1* pDesc,
  IDXGIOutput* pRestrictToOutput,
  IDXGISwapChain1** ppSwapChain );

HRESULT IDXGIFactory2::CreateSwapChainForComposition(
  IUnknown* pDevice,
  const DXGI_SWAP_CHAIN_DESC1* pDesc,
  IDXGIOutput* pRestrictToOutput,
  IDXGISwapChain1** ppSwapChain );

HRESULT IDXGISwapChain::GetDesc(
  DXGI_SWAP_CHAIN_DESC* pDesc );

HRESULT IDXGISwapChain1::GetDesc1(
  DXGI_SWAP_CHAIN_DESC1* pDesc );

HRESULT IDXGISwapChain1::GetHwnd(
  HWND* pHwnd );

HRESULT IDXGISwapChain1::GetCoreWindow(
  REFIID refiid,
  void** ppUnk );
ppUnk: a CoreWindow object

struct DXGI_RGBA{
  float r;
  float g;
  float b;
  float a; };

HRESULT IDXGISwapChain1::SetBackgroundColor(
  const DXGI_RGBA* pColor );

HRESULT IDXGISwapChain1::GetBackgroundColor(
  DXGI_RGBA* pColor );
```

Resizing

```
HRESULT IDXGISwapChain::ResizeBuffers(
  UINT BufferCount,
  UINT Width,
  UINT Height,
  DXGI_FORMAT NewFormat, p19
  UINT SwapChainFlags );
  SwapChainFlags : DXGI_SWAP_CHAIN_FLAG

HRESULT IDXGISwapChain3::ResizeBuffers1(
  UINT BufferCount,
  UINT Width,
  UINT Height,
  DXGI_FORMAT Format, p19
  UINT SwapChainFlags,
  const UINT* pCreationNodeMask,
  IUnknown* const* ppPresentQueue );
  SwapChainFlags : DXGI_SWAP_CHAIN_FLAG

HRESULT IDXGISwapChain::ResizeTarget(
  const DXGI_MODE_DESC* pNewTargetParameters p20 );

HRESULT IDXGISwapChain2::SetSourceSize(
  UINT Width,
  UINT Height );

HRESULT IDXGISwapChain2::GetSourceSize(
  UINT* pWidth,
  UINT* pHeight );
```

Latency control

```
HRESULT IDXGISwapChain2::SetMaximumFrameLatency(
  UINT MaxLatency );

HRESULT IDXGISwapChain2::GetMaximumFrameLatency(
  UINT* pMaxLatency );

HANDLE IDXGISwapChain2::GetFrameLatencyWaitableObject();
```

Full-screen

```
HRESULT IDXGISwapChain::SetFullscreenState(
  BOOL Fullscreen,
  IDXGIOutput* pTarget );

HRESULT IDXGISwapChain::GetFullscreenState(
  BOOL* pFullscreen,
  IDXGIOutput** ppTarget );

HRESULT IDXGISwapChain1::GetFullscreenDesc(
  DXGI_SWAP_CHAIN_FULLSCREEN_DESC* pDesc );
```

Present

```
HRESULT IDXGISwapChain::GetBuffer(
  UINT Buffer,
  REFIID riid,
  void** ppSurface );
ppSurface: a back-buffer

DXGI_PRESENT constants
DXGI_PRESENT_X where X is
  TEST = 0x00000001UL
  DO_NOT_SEQUENCE = 0x00000002UL
  RESTART = 0x00000004UL
  DO_NOT_WAIT = 0x00000008UL
  STEREO_PREFER_RIGHT = 0x00000010UL
  STEREO_TEMPORARY_MONO = 0x00000020UL
  RESTRICT_TO_OUTPUT = 0x00000040UL
  USE_DURATION = 0x00000100UL
  ALLOW_TEARING = 0x00000200UL

HRESULT IDXGISwapChain::Present(
  UINT SyncInterval,
  UINT Flags );
Flags: a combination of DXGI_PRESENT values.

struct DXGI_PRESENT_PARAMETERS{
  UINT DirtyRectsCount;
  RECT* pDirtyRects;
  RECT* pScrollRect;
  POINT* pScrollOffset; };

HRESULT IDXGISwapChain1::Present1(
  UINT SyncInterval,
  UINT PresentFlags,
  const DXGI_PRESENT_PARAMETERS* pPresentParameters );
```

Color Space

```
HRESULT IDXGISwapChain3::CheckColorSpaceSupport(
  DXGI_COLOR_SPACE_TYPE ColorSpace, p19
  UINT* pColorSpaceSupport );
pColorSpaceSupport: a combination of
DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG values.

enum DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG
DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG_X where X is
  PRESENT = 0x1,
  OVERLAY_PRESENT = 0x2

HRESULT IDXGISwapChain3::SetColorSpace1(
  DXGI_COLOR_SPACE_TYPE ColorSpace p19 );
```

Scaling and rotation

```
struct DXGI_MATRIX_3X2_F{
  FLOAT _11;
  FLOAT _12;
  FLOAT _21;
  FLOAT _22;
  FLOAT _31;
  FLOAT _32; };

HRESULT IDXGISwapChain2::SetMatrixTransform(
  const DXGI_MATRIX_3X2_F* pMatrix );

HRESULT IDXGISwapChain2::GetMatrixTransform(
  DXGI_MATRIX_3X2_F* pMatrix );

HRESULT IDXGISwapChain1::SetRotation(
  DXGI_MODE_ROTATION Rotation p20 );

HRESULT IDXGISwapChain1::GetRotation(
  DXGI_MODE_ROTATION* pRotation p20 );
```

Stereo

```
BOOL IDXGIFactory2::IsWindowedStereoEnabled();

HRESULT IDXGIFactory2::RegisterStereoStatusWindow(
  HWND WindowHandle,
  UINT wMsg,
  DWORD* pdwCookie );

HRESULT IDXGIFactory2::RegisterStereoStatusEvent(
  HANDLE hEvent,
  DWORD* pdwCookie );

void IDXGIFactory2::UnregisterStereoStatus(
  DWORD dwCookie );

BOOL IDXGISwapChain1::IsTemporaryMonoSupported();
```

High Dynamic Range and Wide Color Gamut

```
enum DXGI_HDR_METADATA_TYPE
DXGI_HDR_METADATA_TYPE_X where X is
  NONE,
  HDR1

struct DXGI_HDR_METADATA_HDR10{
  UINT16 RedPrimary[ 2 ];
  UINT16 GreenPrimary[ 2 ];
  UINT16 BluePrimary[ 2 ];
  UINT16 WhitePoint[ 2 ];
  UINT MaxMasteringLuminance;
  UINT MinMasteringLuminance;
  UINT16 MaxContentLightLevel;
  UINT16 MaxFrameAverageLightLevel; ;

HRESULT IDXGISwapChain3::SetHDRMetaData(
  DXGI_HDR_METADATA_TYPE Type,
  UINT Size,
  void* pMetaData);
pMetaData: a DXGI_HDR_METADATA_HDR10 record
```

Output

Represents an adapter output (such as a monitor).

Output enumeration

```
HRESULT IDXGIAdapter::EnumOutputs(
  UINT Output,
  IDXGIOutput** ppOutput );
```

Swap Chain output association

```
HRESULT IDXGISwapChain::GetContainingOutput(
  IDXGIOutput** ppOutput );

HRESULT IDXGISwapChain1::GetRestrictToOutput(
  IDXGIOutput** ppRestrictToOutput );
```

Display modes

```
struct DXGI_OUTPUT_DESC{
  WCHAR DeviceName[32];
  RECT DesktopCoordinates;
  BOOL AttachedToDesktop;
  DXGI_MODE_ROTATION Rotation; p20
  HMONITOR Monitor; };
```

```
HRESULT IDXGIOutput::GetDesc(
  DXGI_OUTPUT_DESC* pDesc );

DXGI_ENUM_MODES constants
DXGI_ENUM_MODES_X where X is
  INTERLACED = 1UL
  SCALING = 2UL
  STEREO = 4UL
  DISABLED_STEREO = 8UL

HRESULT IDXGIOutput::GetDisplayModeList(
  DXGI_FORMAT EnumFormat, p19
  UINT Flags,
  UINT* pNumModes,
  DXGI_MODE_DESC* pDesc p20 );
Flags: a combination of DXGI_ENUM_MODES values

HRESULT IDXGIOutput::FindClosestMatchingMode(
  const DXGI_MODE_DESC* pModeToMatch, p20
  DXGI_MODE_DESC* pClosestMatch, p20
  IUnknown* pConcernedDevice );
pConcernedDevice: a Direct3D device, optional. If NULL, only
modes matching pModeToMatch will be returned; only formats
supported for scan-out by the device otherwise

struct DXGI_MODE_DESC1{
  UINT Width;
  UINT Height;
  DXGI_RATIONAL RefreshRate; p20
  DXGI_FORMAT Format; p19
  DXGI_MODE_SCANLINE_ORDER ScanlineOrdering; p20
  DXGI_MODE_SCALING Scaling; p20
  BOOL Stereo; };

HRESULT IDXGIOutput1::GetDisplayModeList1(
  DXGI_FORMAT EnumFormat, p19
  UINT Flags,
  UINT* pNumModes,
  DXGI_MODE_DESC1* pDesc );
Flags: a combination of DXGI_ENUM_MODES values

HRESULT IDXGIOutput1::FindClosestMatchingMode1(
  const DXGI_MODE_DESC1* pModeToMatch,
  DXGI_MODE_DESC1* pClosestMatch,
  IUnknown* pConcernedDevice );
pConcernedDevice: a Direct3D device, optional. If NULL, only
modes matching pModeToMatch will be returned; only formats
supported for scan-out by the device otherwise
```

Colour, gamma and overlay

```
struct DXGI_GAMMA_CONTROL_CAPABILITIES{
  BOOL ScaleAndOffsetSupported;
  float MaxConvertedValue;
  float MinConvertedValue;
  UINT NumGammaControlPoints;
  float ControlPointPositions[1025]; };

HRESULT IDXGIOutput::GetGammaControlCapabilities(
  DXGI_GAMMA_CONTROL_CAPABILITIES* pGammaCaps );

struct DXGI_RGB{
  float Red;
  float Green;
  float Blue; };

struct DXGI_GAMMA_CONTROL{
  DXGI_RGB Scale;
  DXGI_RGB Offset;
  DXGI_RGB GammaCurve[1025]; };

HRESULT IDXGIOutput::SetGammaControl(
  const DXGI_GAMMA_CONTROL* pArray );

HRESULT IDXGIOutput::GetGammaControl(
  DXGI_GAMMA_CONTROL* pArray );

BOOL IDXGIOutput2::SupportsOverlays();

HRESULT IDXGIOutput3::CheckOverlaySupport(
  DXGI_FORMAT EnumFormat, p19
  IUnknown* pConcernedDevice,
  UINT* pFlags );
pFlags: a combination of DXGI_OVERLAY_SUPPORT_FLAG values

enum DXGI_OVERLAY_SUPPORT_FLAG
DXGI_OVERLAY_SUPPORT_FLAG_X where X is
  DIRECT = 0x1,
  SCALING = 0x2

enum DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG
DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG_X where X is
  PRESENT = 0x1

HRESULT IDXGIOutput4::CheckOverlayColorSpaceSupport(
  DXGI_FORMAT Format, p19
  DXGI_COLOR_SPACE_TYPE ColorSpace, p19
  IUnknown* pConcernedDevice,
  UINT* pFlags );
pFlags: a combination of DXGI_OVERLAY_COLOR_SPACE_SUPPORT_
FLAG values
```

Common Types

This section lists types are referenced in multiple places on preceding pages, in alphabetical order.

```
enum D3D_FEATURE_LEVEL
D3D_FEATURE_X where X is
  LEVEL_9_1, N.A.
  LEVEL_9_2, N.A.
  LEVEL_9_3, N.A.
  LEVEL_10_0, N.A.
  LEVEL_10_1, N.A.
  LEVEL_11_0,
  LEVEL_11_1,
  LEVEL_12_0,
  LEVEL_12_1

enum D3D_ROOT_SIGNATURE_VERSION
D3D_ROOT_SIGNATURE_X where X is
  VERSION_1 = 0x1
  VERSION_1_1 = 0x2

CX struct D3D12_CLEAR_VALUE{
  DXGI_FORMAT Format
  union{
    FLOAT Color[4];
    D3D12_DEPTH_STENCIL_VALUE DepthStencil; }; };

enum D3D12_COMMAND_LIST_TYPE
D3D12_COMMAND_LIST_TYPE_X where X is
  DIRECT,
  BUNDLE,
  COMPUTE,
  COPY

enum D3D12_COMPARISON_FUNC
D3D12_COMPARISON_FUNC_X where X is
  NEVER,
  LESS,
  EQUAL,
  LESS_EQUAL,
  GREATER,
  NOT_EQUAL,
  GREATER_EQUAL,
  ALWAYS

CX struct D3D12_CPU_DESCRIPTOR_HANDLE{
  SIZE_T ptr; };

struct D3D12_DEPTH_STENCIL_VALUE{
  FLOAT Depth;
  UINT8 Stencil; };
```

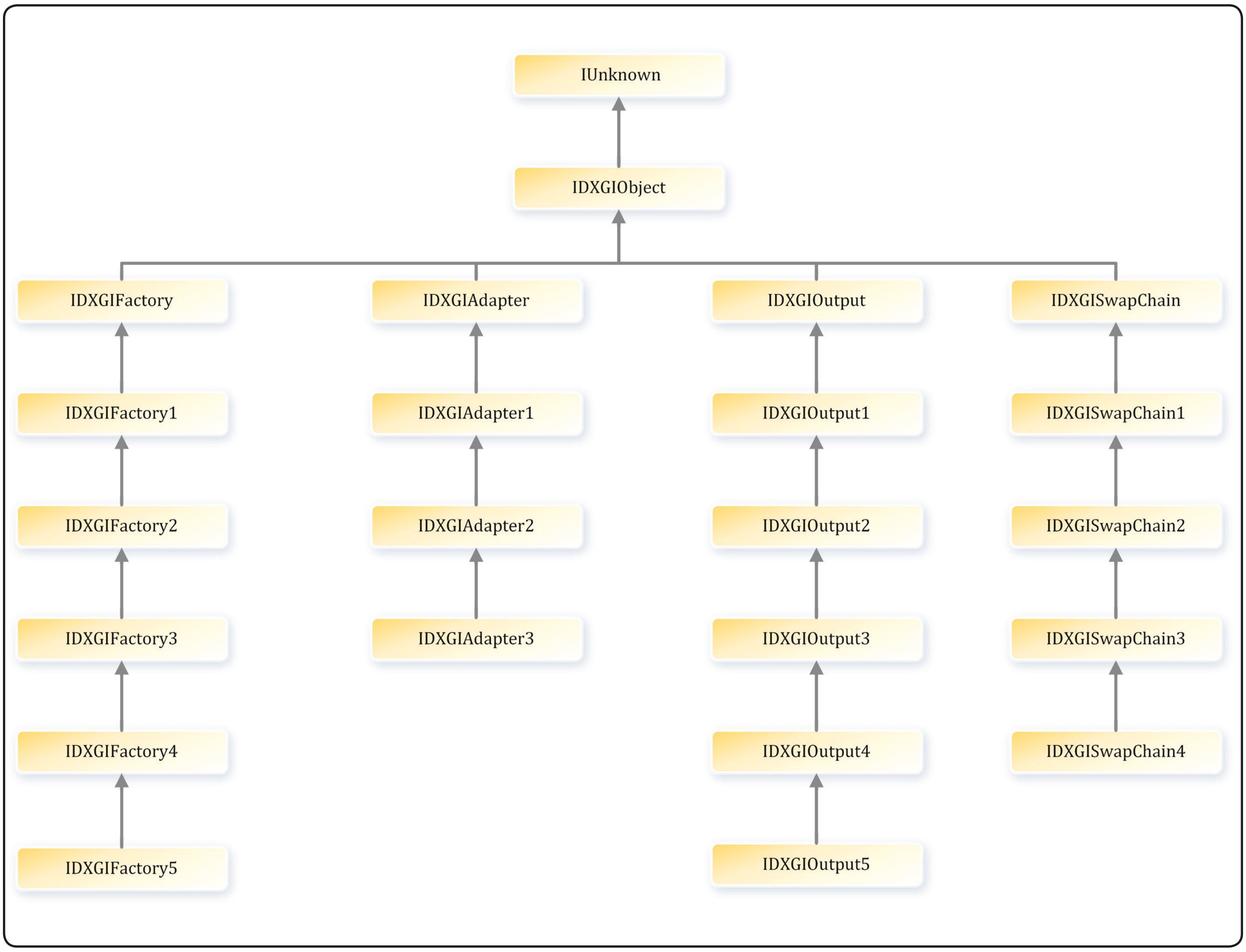
```
enum D3D12_FILTER
D3D12_FILTER_X where X is
  MIN_MAG_MIP_POINT,
  MIN_MAG_POINT_MIP_LINEAR,
  MIN_POINT_MAG_LINEAR_MIP_POINT,
  MIN_POINT_MAG_MIP_LINEAR,
  MIN_LINEAR_MAG_MIP_POINT,
  MIN_LINEAR_MAG_POINT_MIP_LINEAR,
  MIN_MAG_LINEAR_MIP_POINT,
  MIN_MAG_MIP_LINEAR,
  ANISOTROPIC,
  COMPARISON_MIN_MAG_MIP_POINT,
  COMPARISON_MIN_MAG_POINT_MIP_LINEAR,
  COMPARISON_MIN_POINT_MAG_LINEAR_MIP_POINT,
  COMPARISON_MIN_POINT_MAG_MIP_LINEAR,
  COMPARISON_MIN_LINEAR_MAG_MIP_POINT,
  COMPARISON_MIN_LINEAR_MAG_POINT_MIP_LINEAR,
  COMPARISON_MIN_MAG_LINEAR_MIP_POINT,
  COMPARISON_MIN_MAG_MIP_LINEAR,
  COMPARISON_ANISOTROPIC,
  MINIMUM_MIN_MAG_MIP_POINT,
  MINIMUM_MIN_MAG_POINT_MIP_LINEAR,
  MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT,
  MINIMUM_MIN_POINT_MAG_MIP_LINEAR,
  MINIMUM_MIN_LINEAR_MAG_MIP_POINT,
  MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR,
  MINIMUM_MIN_MAG_LINEAR_MIP_POINT,
  MINIMUM_MIN_MAG_MIP_LINEAR,
  MINIMUM_ANISOTROPIC,
  MAXIMUM_MIN_MAG_MIP_POINT,
  MAXIMUM_MIN_MAG_POINT_MIP_LINEAR,
  MAXIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT,
  MAXIMUM_MIN_POINT_MAG_MIP_LINEAR,
  MAXIMUM_MIN_LINEAR_MAG_MIP_POINT,
  MAXIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR,
  MAXIMUM_MIN_MAG_LINEAR_MIP_POINT,
  MAXIMUM_MIN_MAG_MIP_LINEAR,
  MAXIMUM_ANISOTROPIC

enum D3D12_FILTER_REDUCTION_TYPE
D3D12_FILTER_REDUCTION_TYPE_X where X is
  STANDARD,
  COMPARISON,
  MINIMUM,
  MAXIMUM
Used with D3D12 filtering macros

enum D3D12_FILTER_TYPE
D3D12_FILTER_TYPE_X where X is
  POINT,
  LINEAR
Used with D3D12 filtering macros

CX struct D3D12_GPU_DESCRIPTOR_HANDLE{
  UINT64 ptr; };

typedef UINT64 D3D12_GPU_VIRTUAL_ADDRESS;

CX struct D3D12_RECT{
  LONG left;
  LONG top;
  LONG right;
  LONG bottom; }

CX struct D3D12_RESOURCE_DESC{
  D3D12_RESOURCE_DIMENSION Dimension;
  UINT64 Alignment;
  UINT64 Width;
  UINT Height;
  UINT16 DepthOrArraySize;
  UINT16 MipLevels;
  DXGI_FORMAT Format;
  DXGI_SAMPLE_DESC SampleDesc;
  D3D12_TEXTURE_LAYOUT Layout;
  D3D12_RESOURCE_FLAGS Flags; };

enum D3D12_RESOURCE_DIMENSION
D3D12_RESOURCE_DIMENSION_X where X is
  UNKNOWN,
  BUFFER,
  TEXTURE1D,
  TEXTURE2D,
  TEXTURE3D

enum D3D12_RESOURCE_FLAGS
D3D12_RESOURCE_FLAG_X where X is
  NONE = 0,
  ALLOW_RENDER_TARGET = 0x1,
  ALLOW_DEPTH_STENCIL = 0x2,
  ALLOW_UNORDERED_ACCESS = 0x4,
  DENY_SHADER_RESOURCE = 0x8,
  ALLOW_CROSS_ADAPTER = 0x10,
  ALLOW_SIMULTANEOUS_ACCESS = 0x20

enum D3D12_RESOURCE_STATES
D3D12_RESOURCE_STATE_X where X is
  COMMON = 0,
  VERTEX_AND_CONSTANT_BUFFER = 0x1,
  INDEX_BUFFER = 0x2,
  RENDER_TARGET = 0x4,
  UNORDERED_ACCESS = 0x8,
  DEPTH_WRITE = 0x10,
  DEPTH_READ = 0x20,
  NON_PIXEL_SHADER_RESOURCE = 0x40,
  PIXEL_SHADER_RESOURCE = 0x80,
  STREAM_OUT = 0x100,
  INDIRECT_ARGUMENT = 0x200,
  COPY_DEST = 0x400,
  COPY_SOURCE = 0x800,
  RESOLVE_DEST = 0x1000,
  RESOLVE_SOURCE = 0x2000,
  GENERIC_READ = (((((0x1|0x2)|0x40)|0x80)|0x200)|0x800),
  PRESENT = 0,
  PREDICATION = 0x200

enum D3D12_TEXTURE_ADDRESS_MODE
D3D12_TEXTURE_ADDRESS_MODE_X where X is
  WRAP,
  MIRROR,
  CLAMP,
  BORDER,
  MIRROR_ONCE

enum D3D12_TEXTURE_LAYOUT
D3D12_TEXTURE_LAYOUT_X where X is
  UNKNOWN,
  ROW_MAJOR,
  64KB_UNDEFINED_SWIZZLE,
  64KB_STANDARD_SWIZZLE

enum DXGI_COLOR_SPACE_TYPE
DXGI_COLOR_SPACE_X where X is
  RGB_FULL_G22_NONE_P709,
  RGB_FULL_G10_NONE_P709,
  RGB_STUDIO_G22_NONE_P709,
  RGB_STUDIO_G22_NONE_P2020,
  RESERVED,
  YCBCR_FULL_G22_NONE_P709_X601,
  YCBCR_STUDIO_G22_LEFT_P601,
  YCBCR_FULL_G22_LEFT_P601,
  YCBCR_STUDIO_G22_LEFT_P709,
  YCBCR_FULL_G22_LEFT_P709,
  YCBCR_STUDIO_G22_LEFT_P2020,
  YCBCR_FULL_G22_LEFT_P2020,
  RGB_FULL_G2084_NONE_P2020,
  YCBCR_STUDIO_G2084_LEFT_P2020,
  RGB_STUDIO_G2084_NONE_P2020,
  YCBCR_STUDIO_G22_TOPLEFT_P2020,
  YCBCR_STUDIO_G2084_TOPLEFT_P2020,
  RGB_FULL_G22_NONE_P2020,
  CUSTOM

enum DXGI_FORMAT
DXGI_FORMAT_X where X is
  UNKNOWN,
  R32G32B32A32_TYPELESS,
  R32G32B32A32_FLOAT,
  R32G32B32A32_UINT,
  R32G32B32A32_SINT,
  R32G32B32_TYPELESS,
  R32G32B32_FLOAT,
  R32G32B32_UINT,
  R32G32B32_SINT,
  R16G16B16A16_TYPELESS,
  R16G16B16A16_FLOAT,
  R16G16B16A16_UNORM,
  R16G16B16A16_UINT,
  R16G16B16A16_SNORM,
  R16G16B16A16_SINT,
  R32G32_TYPELESS,
  R32G32_FLOAT,
  R32G32_UINT,
  R32G32_SINT,
  R32G8X24_TYPELESS,
  D32_FLOAT_S8X24_UINT,
  R32_FLOAT_X8X24_TYPELESS,
  X32_TYPELESS_G8X24_UINT,
  R10G10B10A2_TYPELESS,
  R10G10B10A2_UNORM,
  R10G10B10A2_UINT,
  R11G11B10_FLOAT,
  R8G8B8A8_TYPELESS,
  R8G8B8A8_UNORM,
  R8G8B8A8_UNORM_SRGB,
  R8G8B8A8_UINT,
  R8G8B8A8_SNORM,
  R8G8B8A8_SINT,
  R16G16_TYPELESS,
  R16G16_FLOAT,
  R16G16_UNORM,
  R16G16_UINT,
  R16G16_SNORM,
  R16G16_SINT,
  R32_TYPELESS,
  D32_FLOAT,
  R32_FLOAT,
  R32_UINT,
  R32_SINT,
  R24G8_TYPELESS,
  D24_UNORM_S8_UINT,
  R24_UNORM_X8_TYPELESS,
  X24_TYPELESS_G8_UINT,
  R8G8_TYPELESS,
  R8G8_UNORM,
  R8G8_UINT,
  R8G8_SNORM,
  R8G8_SINT,
  R16_TYPELESS,
  R16_FLOAT,
  D16_UNORM,
  R16_UNORM,
  R16_UINT,
  R16_SNORM,
  R16_SINT,
  R8_TYPELESS,
  R8_UNORM,
  R8_UINT,
  R8_SNORM,
  R8_SINT,
  A8_UNORM,
  R1_UNORM,
  R9G9B9E5_SHAREDEXP,
  R8G8_B8G8_UNORM,
  G8R8_G8B8_UNORM,
  BC1_TYPELESS,
  BC1_UNORM,
  BC1_UNORM_SRGB,
  BC2_TYPELESS,
  BC2_UNORM,
  BC2_UNORM_SRGB,
  BC3_TYPELESS,
  BC3_UNORM,
  BC3_UNORM_SRGB,
  BC4_TYPELESS,
  BC4_UNORM,
  BC4_SNORM,
  BC5_TYPELESS,
  BC5_UNORM,
  BC5_SNORM,
  B5G6R5_UNORM,
  B5G5R5A1_UNORM,
  B8G8R8A8_UNORM,
  B8G8R8X8_UNORM,
  R10G10B10_XR_BIAS_A2_UNORM,
  B8G8R8A8_TYPELESS,
  B8G8R8A8_UNORM_SRGB,
  B8G8R8X8_TYPELESS,
  B8G8R8X8_UNORM_SRGB,
  BC6H_TYPELESS,
  BC6H_UF16,
  BC6H_SF16,
  BC7_TYPELESS,
  BC7_UNORM,
  BC7_UNORM_SRGB,
  AYUV,
  Y410,
  Y416,
  NV12,
  P010,
  P016,
  420_OPAQUE,
  YUY2,
  Y210,
  Y216,
  NV11,
  AI44,
  IA44,
  P8,
  A8P8,
  B4G4R4A4_UNORM,
  P208,
  V208,
  V408,
  FORCE_UINT N.A.

struct DXGI_MODE_DESC{
  UINT Width;
  UINT Height;
  DXGI_RATIONAL RefreshRate;
  DXGI_FORMAT Format;
  DXGI_MODE_SCANLINE_ORDER ScanlineOrdering;
  DXGI_MODE_SCALING Scaling; };

enum DXGI_MODE_ROTATION
DXGI_MODE_ROTATION_X where X is
  UNSPECIFIED,
  IDENTITY,
  ROTATE90,
  ROTATE180,
  ROTATE270

struct DXGI_RATIONAL{
  UINT Numerator;
  UINT Denominator; };

struct DXGI_SAMPLE_DESC{
  UINT Count;
  UINT Quality; };
Quality: one of the following values
  DXGI_STANDARD_MULTISAMPLE_QUALITY_PATTERN,
  DXGI_CENTER_MULTISAMPLE_QUALITY_PATTERN

enum DXGI_MODE_SCALING
DXGI_MODE_SCALING_X where X is
  UNSPECIFIED,
  CENTERED,
  STRETCHED

enum DXGI_MODE_SCANLINE_ORDER
DXGI_MODE_SCANLINE_ORDER_X
  UNSPECIFIED,
  PROGRESSIVE,
  UPPER_FIELD_FIRST,
  LOWER_FIELD_FIRST

interface ID3DBlob : IUnknown{
  LPVOID GetBufferPointer();
  SIZE_T GetBufferSize(); };
```

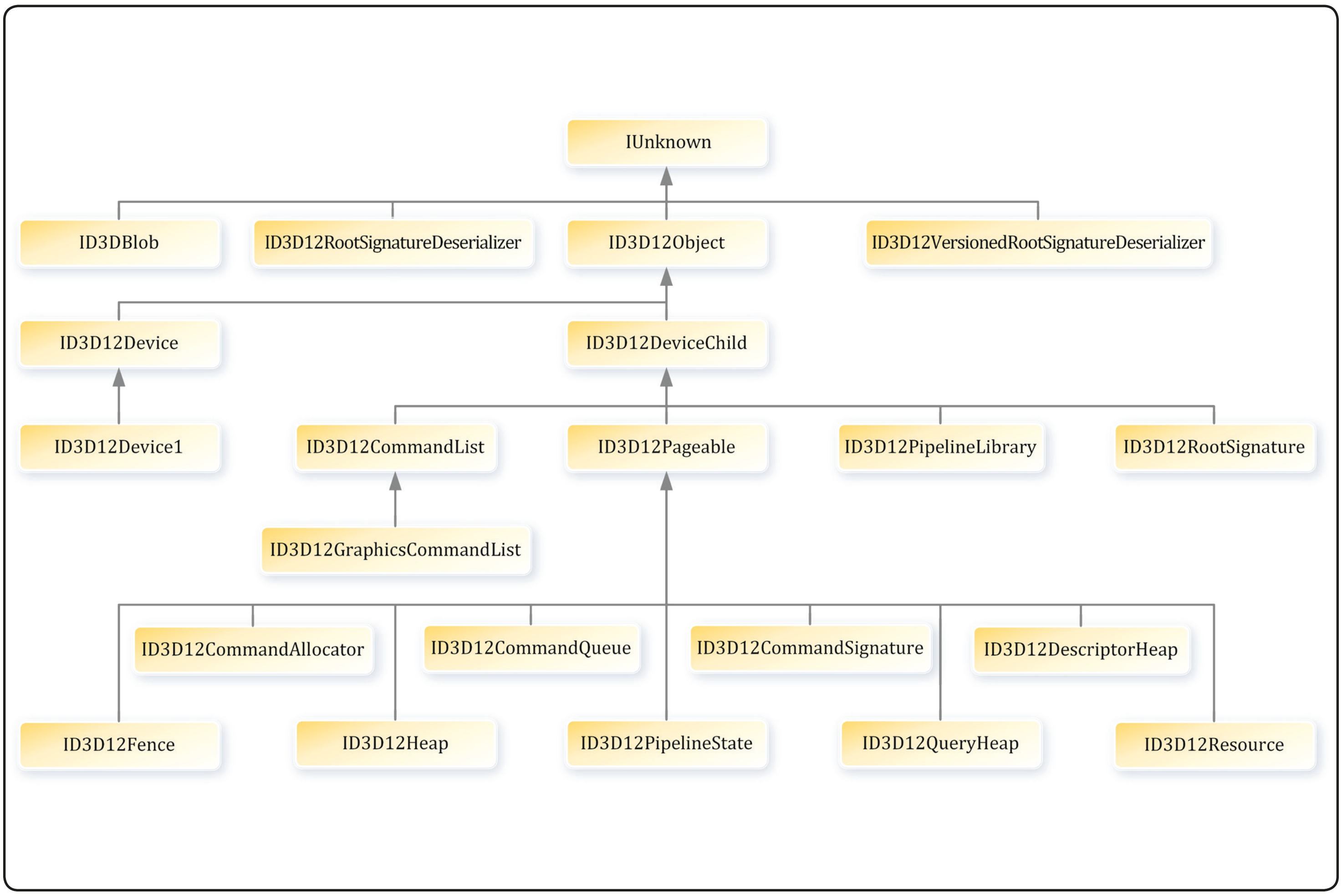
Usefull Links and Resources

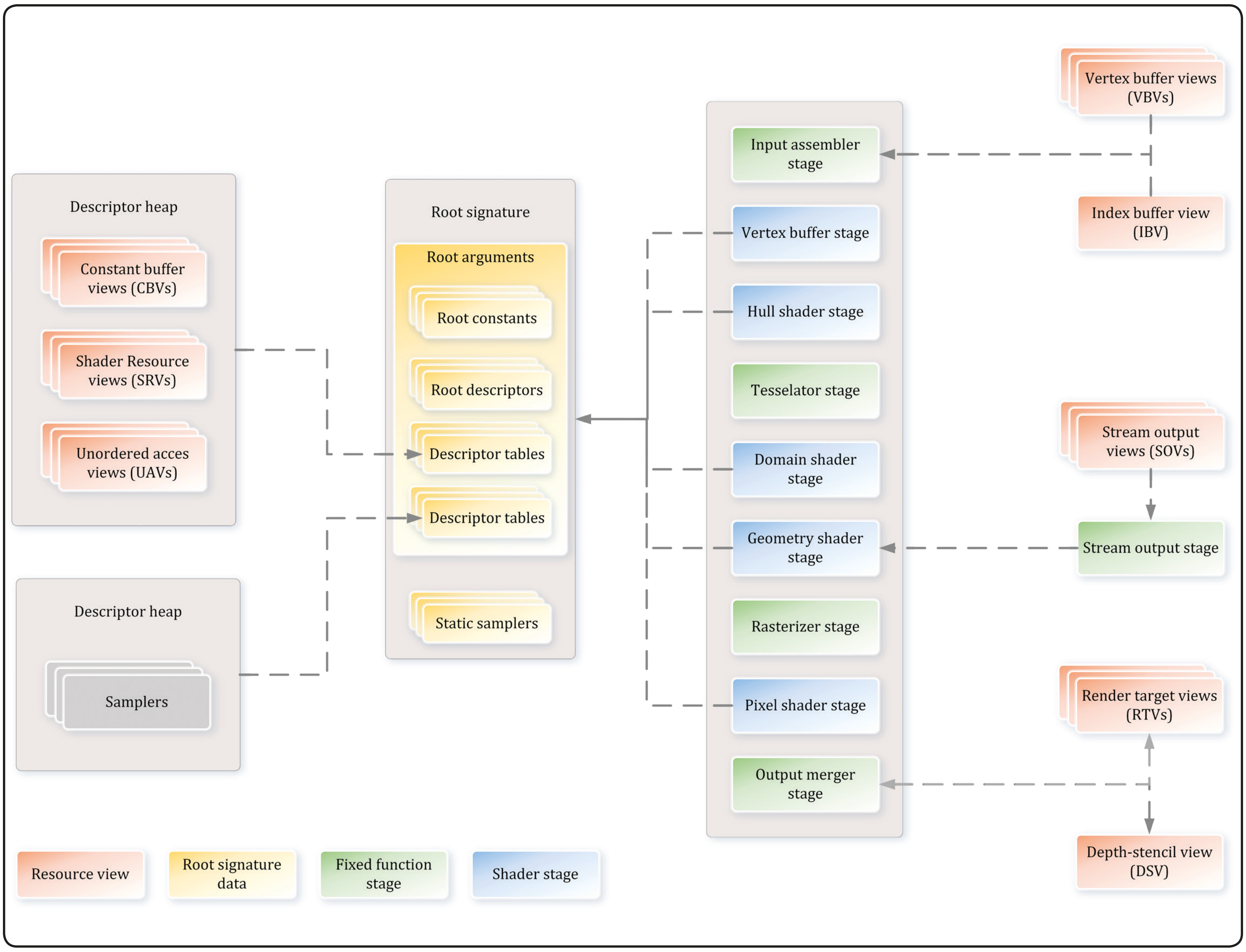
Below follows a list of useful links and resources about Direct3D 12 programming.

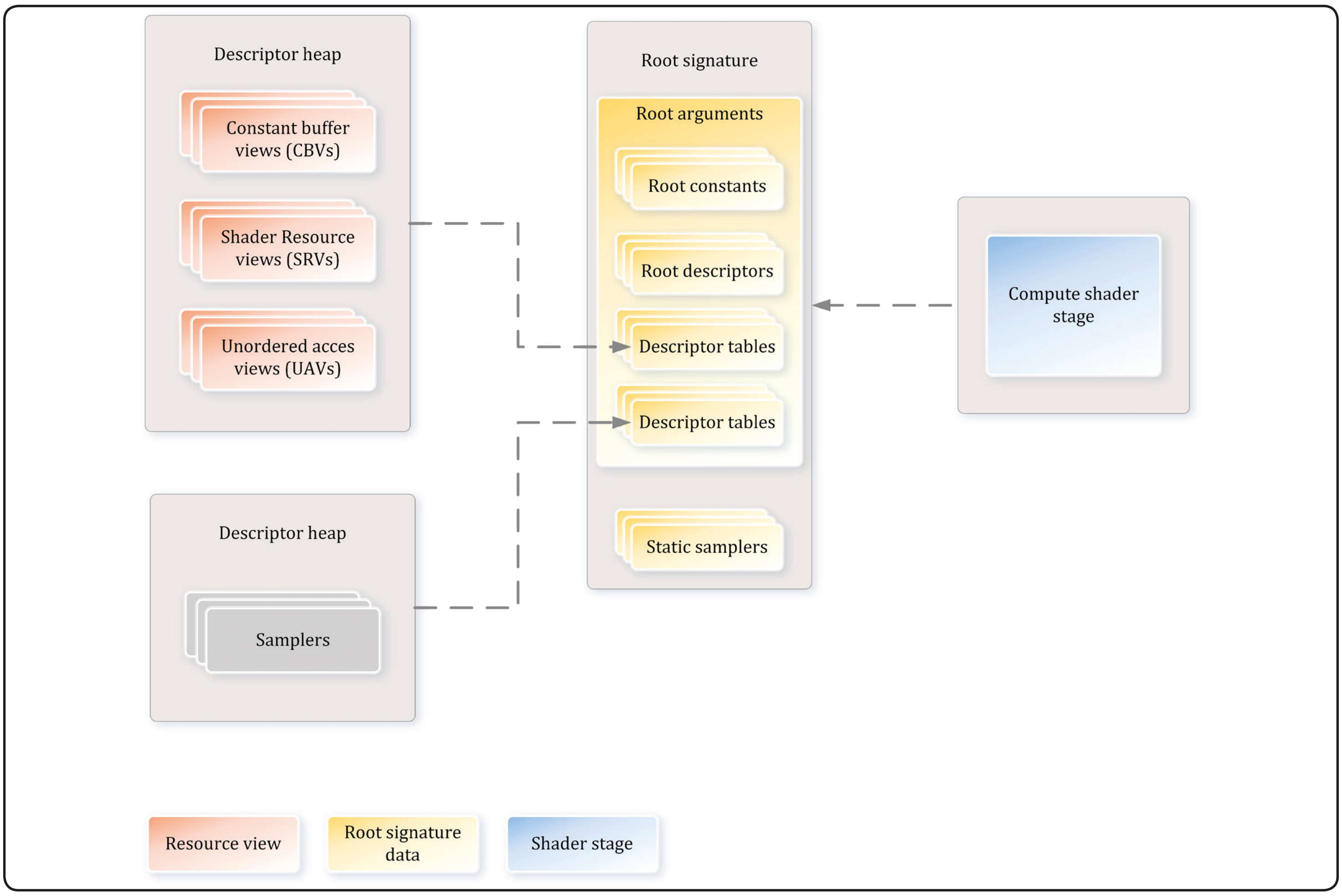
- Microsoft DirectX Graphics Developer Hub: *www.directxtech.com/*
- Microsoft DirectX Graphics samples: *github.com/Microsoft/DirectX-Graphics-Samples*
- Microsoft DirectX 12 and Graphics Education Youtube channel: *www.youtube.com/channel/UCiaX2B8XiXR70jaN7NK-FpA*
- Microsoft DirectX MSDN *blog: blogs.msdn.com/b/directx/*
- Chuck Walbourn - MSFT - Games for Windows and the DirectX SDK blog: *blogs.msdn.microsoft.com/chuckw/*
- DirectXTK12 - DirectX Tool Kit for DirectX 12: *github.com/Microsoft/DirectXTK12*
- DirectXMesh - DirectX Mesh Library: *github.com/Microsoft/DirectXMesh*
- DirectXTex - DirectX Texture Library: *github.com/Microsoft/DirectXTex*
- Intel Graphics Performance Analyzers: *software.intel.com/en-us/gpa*
- Intel Asteroids DirectX 12 sample: *github.com/GameTechDev/asteroids_d3d12*
- Intel flip-model swap chain interactive sample: *software.intel.com/en-us/articles/sample-application-for-direct3d-12-flip-model-swap-chains*
- Intel DirectX 12 explicit multi-adapter sample: *github.com/GameTechDev/DX12-Multi-Adapter*
- NVIDIA Nsight: *www.nvidia.com/object/nsight.html*
- NVIDIA DX12 Do's And Don'ts: *developer.nvidia.com/dx12-dos-and-donts*
- AMD CodeXL: *gpuopen.com/compute-product/codexl/*
- AMD GPU PerfStudio: *developer.amd.com/tools-and-sdks/graphics-development/gpu-perfstudio/*
- AMD GPUOpen DirectX 12 posts: *gpuopen.com/tag/dx12/*
- AMD Hello D3D12 introductory samples: *github.com/GPUOpen-LibrariesAndSDKs/HelloD3D12*
- AMD "async compute" sample: *github.com/GPUOpen-LibrariesAndSDKs/nBodyD3D12/tree/master/Samples/D3D12nBodyGravity*
- Kevin Örtegren, DirectX 12 light culling technique featured in GPU Pro 7: *github.com/kevinortegren/ClusteredShadingConservative*
- Matt Pettineo, Bindless Texturing for Deferred Rendering and Decals: *github.com/TheRealMJP/DeferredTexturing*
- SharpDX: *sharpdx.org/*

Additional Notes


IUnknown
IDXGIObject
IDXGIFactory
IDXGIFactory1
IDXGIFactory2
IDXGIFactory3
IDXGIFactory4
IDXGIFactory5
IDXGIAdapter
IDXGIAdapter1
IDXGIAdapter2
IDXGIAdapter3
IDXGIOutput
IDXGIOutput1
IDXGIOutput2
IDXGIOutput3
IDXGIOutput4
IDXGIOutput5
IDXGISwapChain
IDXGISwapChain1
IDXGISwapChain2
IDXGISwapChain3
IDXGISwapChain4


IUnknown
ID3DBlob
ID3D12RootSignatureDeserializer
ID3D12Object
ID3D12VersionedRootSignatureDeserializer
ID3D12Device
ID3D12DeviceChild
ID3D12Device1
ID3D12CommandList
ID3D12Pageable
ID3D12PipelineLibrary
ID3D12RootSignature
ID3D12GraphicsCommandList
ID3D12CommandAllocator
ID3D12CommandQueue
ID3D12CommandSignature
ID3D12DescriptorHeap
ID3D12Fence
ID3D12Heap
ID3D12PipelineState
ID3D12QueryHeap
ID3D12Resource


Descriptor heap
Constant buffer views (CBVs)
Shader Resource views (SRVs)
Unordered acces views (UAVs)
Descriptor heap
Samplers
Root signature
Root arguments
Root constants
Root descriptors
Descriptor tables
Descriptor tables
Static samplers
Input assembler stage
Vertex buffer stage
Hull shader stage
Tesselator stage
Domain shader stage
Geometry shader stage
Rasterizer stage
Pixel shader stage
Output merger stage
Vertex buffer views (VBVs)
Index buffer view (IBV)
Stream output views (SOVs)
Stream output stage
Render target views (RTVs)
Depth-stencil view (DSV)
Resource view
Root signature data
Fixed function stage
Shader stage


Descriptor heap
Constant buffer views (CBVs)
Shader Resource views (SRVs)
Unordered acces views (UAVs)
Descriptor heap
Samplers
Root signature
Root arguments
Root constants
Root descriptors
Descriptor tables
Descriptor tables
Static samplers
Compute shader stage
Resource view
Root signature data
Shader stage