

Contents

1	API Reference	1
1.1	Context	2
1.2	Geometry Group	62
1.3	Group Node	78
1.4	Selector Node	92
1.5	Transform Node	116
1.6	Acceleration Structure	129
1.7	Geometry Instance	152
1.8	Geometry	176
1.9	Material	200
1.10	Program	223
1.11	Buffer	239
1.12	Texture Sampler	288
1.13	Variables	338
1.14	Context Free Functions	365
2	CUDA C Reference	377
2.1	Declarations	378
2.2	Types	385
2.3	Functions	397
3	Appendix	413
3.1	Interop Formats	414

Chapter 1

API Reference

1.1 Context

NAME

Context

DESCRIPTION

This section describes the API functions for creation and handling of rendering contexts.

rtContextCompile

rtContextCreate

rtContextDeclareVariable

rtContextDestroy

rtContextGetAttribute

rtContextGetDevices

rtContextGetDeviceCount

rtContextGetEntryPointCount

rtContextGetErrorString

rtContextGetExceptionEnabled

rtContextGetExceptionProgram

rtContextGetMissProgram

rtContextGetPrintBufferSize

rtContextGetPrintEnabled

rtContextGetPrintLaunchIndex

rtContextGetRayGenerationProgram

rtContextGetRayTypeCount

rtContextGetRunningState

rtContextGetStackSize

rtContextGetVariableCount

rtContextGetVariable

rtContextQueryVariable

rtContextRemoveVariable

rtContextSetAttribute

rtContextSetD3D9Device

rtContextSetD3D10Device
rtContextSetD3D11Device
rtContextSetDevices
rtContextSetEntryPointCount
rtContextSetExceptionEnabled
rtContextSetExceptionProgram
rtContextSetMissProgram
rtContextSetPrintBufferSize
rtContextSetPrintEnabled
rtContextSetPrintLaunchIndex
rtContextSetRayGenerationProgram
rtContextSetRayTypeCount
rtContextSetStackSize
rtContextSetTimeoutCallback
rtContextLaunch
rtContextValidate

HISTORY

Contexts were introduced in *OptiX* 1.0.

SEE ALSO

Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.1.1 `rtContextCompile`

NAME

`rtContextCompile` - Compiles a context object.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextCompile(RTcontext context)
```

PARAMETERS

`context`

The context to be compiled.

DESCRIPTION

`rtContextCompile` creates a final computation kernel from the given context's programs and scene hierarchy. This kernel will be executed upon subsequent invocations of `rtContextLaunch`.

Calling `rtContextCompile` is not strictly necessary since any changes to the scene specification or programs will cause an internal compilation upon the next `rtContextLaunch` call. `rtContextCompile` allows the application to control when the compilation work occurs.

Conversely, if no changes to the scene specification or programs have occurred since the last compilation, `rtContextCompile` and `rtContextLaunch` will not perform a recompilation.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_INVALID_SOURCE`

HISTORY

`rtContextCompile` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextLaunch

1.1.2 `rtContextCreate`

NAME

`rtContextCreate` - Creates a new context object.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextCreate(RTcontext* context)
```

PARAMETERS

`context`

Handle to context for return value.

DESCRIPTION

`rtContextCreate` allocates and returns a handle to a new context object. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_NO_DEVICE`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextCreate` was introduced in *OptiX* 1.0.

SEE ALSO

1.1.3 rtContextDeclareVariable

NAME

rtContextDeclareVariable - Declares a new named variable associated with this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextDeclareVariable(RTcontext context,
                                  const char* name,
                                  RTvariable* v)
```

PARAMETERS

context

The context node to which the variable will be attached.

name

The name that identifies the variable to be queried.

v

Pointer to variable handle used to return the new object.

DESCRIPTION

rtContextDeclareVariable - Declares a new variable named **name** and associated with this context. Only a single variable of a given name can exist for a given context and any attempt to create multiple variables with the same name will cause a failure with a return value of `RT_ERROR_VARIABLE_REDECLARED`. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer. Return `RT_ERROR_ILLEGAL_SYMBOL` if **name** is not syntactically valid.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_VARIABLE_REDECLARED`

HISTORY

rtContextDeclareVariable was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryDeclareVariable, *rtGeometryInstanceDeclareVariable*, *rtMaterialDeclareVariable*, *rtProgramDeclareVariable*, *rtSelectorDeclareVariable*, *rtContextGetVariable*, *rtContextGetVariableCount*, *rtContextQueryVariable*, *rtContextRemoveVariable*

1.1.4 rtContextDestroy

NAME

rtContextDestroy - Destroys a context and frees all associated resources

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextDestroy(RTcontext context)
```

PARAMETERS

context

Handle of the context to destroy

DESCRIPTION

rtContextDestroy frees all resources, including *OptiX* objects, associated with this object. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL context. `RT_ERROR_LAUNCH_FAILED` may be returned if a previous call to `rtContextLaunch` failed.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_LAUNCH_FAILED`

HISTORY

rtContextDestroy was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate

1.1.5 rtContextGetAttribute

NAME

rtContextGetAttribute - returns an attribute specific to an *OptiX* context.

SYNOPSIS

```
#include <optix.h>

RTresult RTAPI rtContextGetAttribute(RTcontext context,
                                     RTcontextattribute attrib,
                                     RTsize size,
                                     void* p);
```

PARAMETERS

context

The context object to be queried.

attrib

Attribute to query.

size

Size of the attribute being queried. Parameter **p** must have at least this much memory backing it.

p

Return pointer where the value of the attribute will be copied into. This must point to at least **size** bytes of memory.

DESCRIPTION

rtContextGetAttribute() returns in **p** the value of the per context attribute specified by **attrib**.

Each attribute can have a different size. The sizes are given in the following list:

RT_CONTEXT_ATTRIBUTE_MAX_TEXTURE_COUNT	sizeof(int)
RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS	sizeof(int)
RT_CONTEXT_ATTRIBUTE_USED_HOST_MEMORY	sizeof(RTsize)
RT_CONTEXT_ATTRIBUTE_GPU_PAGING_ACTIVE	sizeof(int)
RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF	sizeof(int)
RT_CONTEXT_ATTRIBUTE_AVAILABLE_DEVICE_MEMORY	sizeof(RTsize)

RT_CONTEXT_ATTRIBUTE_MAX_TEXTURE_COUNT queries the maximum number of textures handled by *OptiX*. For *OptiX* versions below 2.5 this value depends on the number of textures supported by CUDA.

RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS queries the number of host CPU threads *OptiX* can use for various tasks.

RT_CONTEXT_ATTRIBUTE_USED_HOST_MEMORY queries the amount of host memory allocated by *OptiX*.

RT_CONTEXT_ATTRIBUTE_GPU_PAGING_ACTIVE queries if software paging of device memory has been turned on by the context. The returned value is a boolean, where 1 means that paging is currently active.

RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF queries if software paging has been prohibited by the user. The returned value is a boolean, where 0 means that *OptiX* is allowed to activate paging if necessary, 1 means that paging is always off.

RT_CONTEXT_ATTRIBUTE_AVAILABLE_DEVICE_MEMORY queries the amount of free device memory.

Some attributes are used to get per device information. In contrast to **rtDeviceGetAttribute**, these attributes are determined by the context and are therefore queried through the context. This is done by summing the attribute with the *OptiX* ordinal number when querying the attribute. The following are per device attributes.

RT_CONTEXT_ATTRIBUTE_AVAILABLE_DEVICE_MEMORY

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE - Can be returned if **size** does not match the proper size of the attribute, if **p** is NULL, or if **attribute+ordinal** does not correspond to an *OptiX* device.

HISTORY

rtContextGetAttribute was introduced in *OptiX* 2.0.

SEE ALSO

rtContextGetDeviceCount, *rtContextSetAttribute*, *rtDeviceGetAttribute*

1.1.6 `rtContextGetDevices`

NAME

`rtContextGetDevices` - Retrieve a list of hardware devices being used by the kernel.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetDevices(RTcontext context,
                             int* devices)
```

PARAMETERS

`context`

The context to which the hardware list is applied.

`devices`

Return parameter for the list of devices. The memory must be able to hold entries numbering least the number of devices as returned by `rtContextGetDeviceCount`.

DESCRIPTION

`rtContextGetDevices` retrieves a list of hardware devices used during execution of the subsequent trace kernels.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetDevices` was introduced in *OptiX 2.0*.

SEE ALSO

rtContextSetDevices, *rtContextGetDeviceCount*

1.1.7 `rtContextGetDeviceCount`

NAME

`rtContextGetDeviceCount` - Query the number of devices currently being used.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetDeviceCount(RTcontext context,
                                unsigned int* count)
```

PARAMETERS

`context`

The context containing the devices.

`devices`

Return parameter for the device count.

DESCRIPTION

`rtContextGetDeviceCount` - Query the number of devices currently being used.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetDeviceCount` was introduced in *OptiX 2.0*.

SEE ALSO

rtContextSetDevices, *rtContextGetDevices*

1.1.8 `rtContextGetEntryPointCount`

NAME

`rtContextGetEntryPointCount` - Query the number of entry points for this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetEntryPointCount(RTcontext context,
                                     unsigned int* num_entry_points)
```

PARAMETERS

`context`

The context node to be queried.

`num_entry_points`

Return parameter for passing back the entry point count.

DESCRIPTION

`rtContextGetEntryPointCount` passes back the number of entry points associated with this context in `num_entry_points`. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetEntryPointCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetEntryPointCount

1.1.9 rtContextGetErrorString

NAME

rtContextGetErrorString - returns the error string associated with a given error.

SYNOPSIS

```
#include <optix.h>

void rtContextGetErrorString(RTcontext context,
                             RTResult code,
                             char** return_string)
```

PARAMETERS

context

The context object to be queried, or NULL.

code

The error code to be converted to string.

return_string

The return parameter for the error string.

DESCRIPTION

rtContextGetErrorString return a descriptive string given an error code. If **context** is valid and additional information is available from the last *OptiX* failure, it will be appended to the generic error code description. **return_string** will be set to point to this string. The memory **return_string** points to will be valid until the next API call that returns a string.

RETURN VALUES

rtContextGetErrorString does not return a value.

HISTORY

rtContextGetErrorString was introduced in *OptiX* 1.0.

SEE ALSO

1.1.10 `rtContextGetExceptionEnabled`

NAME

`rtContextGetExceptionEnabled` - Query whether a specified exception is enabled.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextGetExceptionEnabled(RTcontext context,  
                                             RTexception exception,  
                                             int* enabled)
```

PARAMETERS

`context`

The context to be queried.

`exception`

The exception of which to query the state.

`enabled`

Return parameter to store whether the exception is enabled.

DESCRIPTION

`rtContextGetExceptionEnabled` passes back 1 in the location pointed to by `enabled` if the given exception is enabled, 0 otherwise. `exception` specifies the type of exception to be queried. For a list of available types, see `rtContextSetExceptionEnabled`. If `exception` is `RT_EXCEPTION_ALL`, `enabled` is set to 1 only if all possible exceptions are enabled.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetExceptionEnabled` was introduced in *OptiX* 1.1.

SEE ALSO

rtContextSetExceptionEnabled, rtContextSetExceptionProgram, rtContextGetExceptionProgram, rtGetExceptionCode, rtThrow, rtPrintExceptionDetails

1.1.11 `rtContextGetExceptionProgram`

NAME

`rtContextGetExceptionProgram` - Queries the exception program associated with the given context and entry point.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextGetExceptionProgram(RTcontext context,  
                                     unsigned int entry_point_index,  
                                     RTprogram* program)
```

PARAMETERS

`context`

The context node associated with the exception program.

`entry_point_index`

The entry point index for the desired exception program.

`program`

Return parameter to store the exception program.

DESCRIPTION

`rtContextGetExceptionProgram` passes back the exception program associated with the given context and entry point. This program is set via `rtContextSetExceptionProgram`. Returns `RT_ERROR_INVALID_VALUE` if given an invalid entry point index or NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetExceptionProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetExceptionProgram, rtContextSetEntryPointCount, rtContextSetExceptionEnabled, rtContextGetExceptionEnabled, rtGetExceptionCode, rtThrow, rtPrintExceptionDetails

1.1.12 `rtContextGetMissProgram`

NAME

`rtContextGetMissProgram` - Queries the miss program associated with the given context and ray type.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetMissProgram(RTcontext context,
                                unsigned int ray_type_index,
                                RTprogram* program)
```

PARAMETERS

context

The context node associated with the miss program.

ray_type_index

The ray type index for the desired miss program.

program

Return parameter to store the miss program.

DESCRIPTION

`rtContextGetMissProgram` passes back the miss program associated with the given context and ray type. This program is set via **`rtContextSetMissProgram`**. Returns `RT_ERROR_INVALID_VALUE` if given a NULL pointer or `ray_type_index` is outside of the range `[0, rtContextGetRayTypeCount()-1]`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetMissProgram` was introduced in *OptiX* 1.0.

SEE ALSO

`rtContextSetMissProgram`, *`rtContextGetRayTypeCount`*

1.1.13 rtContextGetPrintBufferSize

NAME

rtContextGetPrintBufferSize - Get the current size of the print buffer.

SYNOPSIS

```
#include <optix.h>

RTresult RTAPI rtContextGetPrintBufferSize(RTcontext context,
                                           RTsize* buffer_size_bytes)
```

PARAMETERS

context

The context from which to query the print buffer size.

buffer_size_bytes

The returned print buffer size in bytes.

DESCRIPTION

rtContextGetPrintBufferSize is used to query the buffer size available to hold data generated by **rtPrintf**. Returns **RT_ERROR_INVALID_VALUE** if passed a NULL pointer.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtContextGetPrintBufferSize was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextSetPrintEnabled*, *rtContextGetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextSetPrintLaunchIndex*, *rtContextGetPrintLaunchIndex*

1.1.14 `rtContextGetPrintEnabled`

NAME

`rtContextGetPrintEnabled` - Query whether text printing from programs is enabled.

SYNOPSIS

```
#include <optix.h>

RTresult RTAPI rtContextGetPrintEnabled(RTcontext context,
                                        int* enabled)
```

PARAMETERS

`context`

The context to be queried.

`enabled`

Return parameter to store whether printing is enabled.

DESCRIPTION

`rtContextGetPrintEnabled` passes back 1 if text printing from programs through `rtPrintf` is currently enabled for this context; 0 otherwise. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetPrintEnabled` was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextSetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextGetPrintBufferSize*, *rtContextSetPrintLaunchIndex*, *rtContextGetPrintLaunchIndex*

1.1.15 `rtContextGetPrintLaunchIndex`

NAME

`rtContextGetPrintLaunchIndex` - Gets the active print launch index.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextGetPrintLaunchIndex(RTcontext context,
                                             int* x,
                                             int* y,
                                             int* z)
```

PARAMETERS

`context`

The context from which to query the print launch index.

`x`

Returns the launch index in the x dimension to which the output of `rtPrintf` invocations is limited. Will not be written to if a NULL pointer is passed.

`y`

Returns the launch index in the y dimension to which the output of `rtPrintf` invocations is limited. Will not be written to if a NULL pointer is passed.

`z`

Returns the launch index in the z dimension to which the output of `rtPrintf` invocations is limited. Will not be written to if a NULL pointer is passed.

DESCRIPTION

`rtContextGetPrintLaunchIndex` is used to query for which launch indices `rtPrintf` generates output. The initial value of (x,y,z) is (-1,-1,-1), which generates output for all indices.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetPrintLaunchIndex` was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextGetPrintEnabled*, *rtContextSetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextGetPrintBufferSize*, *rtContextSetPrintLaunchIndex*

1.1.16 `rtContextGetRayGenerationProgram`

NAME

`rtContextGetRayGenerationProgram` - Queries the ray generation program associated with the given context and entry point.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetRayGenerationProgram(RTcontext context,
                                         unsigned int entry_point_index,
                                         RTprogram* program)
```

PARAMETERS

context

The context node associated with the ray generation program.

entry_point_index

The entry point index for the desired ray generation program.

program

Return parameter to store the ray generation program.

DESCRIPTION

`rtContextGetRayGenerationProgram` passes back the ray generation program associated with the given context and entry point. This program is set via **`rtContextSetRayGenerationProgram`**. Returns `RT_ERROR_INVALID_VALUE` if given an invalid entry point index or `NULL` pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetRayGenerationProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetRayGenerationProgram

1.1.17 `rtContextGetRayTypeCount`

NAME

`rtContextGetRayTypeCount` - Query the number of ray types associated with this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetRayTypeCount(RTcontext context,
                                  unsigned int* num_ray_types)
```

PARAMETERS

`context`

The context node to be queried.

`num_ray_types`

Return parameter to store the number of ray types.

DESCRIPTION

`rtContextGetRayTypeCount` passes back the number of entry points associated with this context in `num_ray_types`. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetRayTypeCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetRayTypeCount

1.1.18 `rtContextGetRunningState`

NAME

`rtContextGetRunningState` - Query whether the given context is currently running.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetRunningState(RTcontext context,
                                  int* running)
```

PARAMETERS

`context`

The context node to be queried.

`running`

Return parameter to store the running state.

DESCRIPTION

`rtContextGetRunningState` passes back 1 if an `rtContextLaunch` is currently active for this context; 0 otherwise. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetRunningState` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextLaunch1D, *rtContextLaunch2D*, *rtContextLaunch3D*

1.1.19 `rtContextGetStackSize`

NAME

`rtContextGetStackSize` - Query the stack size for this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetStackSize(RTcontext context,
                               RTsize* stack_size_bytes)
```

PARAMETERS

`context`

The context node to be queried.

`stack_size_bytes`

Return parameter to store the size of the stack.

DESCRIPTION

`rtContextGetStackSize` passes back the stack size associated with this context in `stack_size_bytes`. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetStackSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetStackSize

1.1.20 `rtContextGetVariableCount`

NAME

`rtContextGetVariableCount` - Returns the number of variables associated with this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetVariableCount(RTcontext context,
                                   unsigned int* count)
```

PARAMETERS

`context`

The context to be queried for number of attached variables.

`count`

Return parameter to store the number of variables.

DESCRIPTION

`rtContextGetVariableCount` returns the number of variables that are currently attached to `context`. Returns `RT_ERROR_INVALID_VALUE` if passed a NULL pointer.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetVariableCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGetVariableCount, *rtGeometryInstanceGetVariableCount*, *rtMaterialGetVariableCount*, *rtProgramGetVariableCount*, *rtSelectorGetVariable*, *rtContextDeclareVariable*, *rtContextGetVariable*, *rtContextQueryVariable*, *rtContextRemoveVariable*

1.1.21 `rtContextGetVariable`

NAME

`rtContextGetVariable` - Queries an indexed variable associated with this context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextGetVariable(RTcontext context,
                             unsigned int index,
                             RTvariable* v)
```

PARAMETERS

`context`

The context node to be queried for an indexed variable.

`index`

The index that identifies the variable to be queried.

`v`

Return value to store the queried variable.

DESCRIPTION

`rtContextGetVariable` queries the variable at position `index` in the variable array from `context` and stores the result in the parameter `v`. A variable has to be declared first with `rtContextDeclareVariable` and `index` has to be in the range `[0, rtContextGetVariableCount()-1]`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextGetVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGetVariable, *rtGeometryInstanceGetVariable*, *rtMaterialGetVariable*, *rtProgramGetVariable*, *rtSelectorGetVariable*, *rtContextDeclareVariable*, *rtContextGetVariableCount*, *rtContextQueryVariable*, *rtContextRemoveVariable*

1.1.22 `rtContextQueryVariable`

NAME

`rtContextQueryVariable` - Returns a named variable associated with this context.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextQueryVariable(RTcontext context,  
                                const char* name,  
                                RTvariable* v)
```

PARAMETERS

`context`

The context node to query a variable from.

`name`

The name that identifies the variable to be queried.

`v`

Return value to store the queried variable.

DESCRIPTION

`rtContextQueryVariable` queries a variable identified by the string `name` from `context` and stores the result in the parameter `v`. A variable has to be declared first with `rtContextDeclareVariable` before it can be queried. The return parameter `v` will be set to 0 if no variable exists with the given name. `RT_ERROR_INVALID_VALUE` will be returned if `name` is `NULL`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextQueryVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryQueryVariable, *rtGeometryInstanceQueryVariable*, *rtMaterialQueryVariable*, *rtProgramQueryVariable*, *rtSelectorQueryVariable*, *rtContextDeclareVariable*, *rtContextGetVariableCount*, *rtContextGetVariable*, *rtContextRemoveVariable*

1.1.23 `rtContextRemoveVariable`

NAME

`rtContextRemoveVariable` - Removes a variable from the given context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextRemoveVariable(RTcontext context,
                                RTvariable v)
```

PARAMETERS

`context`

The context node from which to remove a variable.

`v`

The variable to be removed.

DESCRIPTION

`rtContextRemoveVariable` removes variable `v` from `context` if present. Returns `RT_ERROR_VARIABLE_NOT_FOUND` if the variable is not attached to this context. Returns `RT_ERROR_INVALID_VALUE` if passed an invalid variable.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtContextRemoveVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryRemoveVariable, *rtGeometryInstanceRemoveVariable*, *rtMaterialRemoveVariable*, *rtProgramRemoveVariable*, *rtSelectorRemoveVariable*, *rtContextDeclareVariable*, *rtContextGetVariable*, *rtContextGetVariableCount*, *rtContextQueryVariable*,

1.1.24 `rtContextSetAttribute`

NAME

`rtContextSetAttribute` - set an attribute specific to an *OptiX* context.

SYNOPSIS

```
#include <optix.h>

RTresult RTAPI rtContextSetAttribute(RTcontext context,
                                     RTcontextattribute attrib,
                                     RTsize size,
                                     void* p);
```

PARAMETERS

context

The context object to be modified.

attrib

Attribute to set.

size

Size of the attribute being set.

p

Pointer to where the value of the attribute will be copied from. This must point to at least **size** bytes of memory.

DESCRIPTION

`rtContextSetAttribute()` sets **p** as the value of the per context attribute specified by **attrib**.

Each attribute can have a different size. The sizes are given in the following list:

<code>RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS</code>	<code>sizeof(int)</code>
<code>RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF</code>	<code>sizeof(int)</code>

`RT_CONTEXT_ATTRIBUTE_CPU_NUM_THREADS` sets the number of host CPU threads *OptiX* can use for various tasks.

`RT_CONTEXT_ATTRIBUTE_GPU_PAGING_FORCED_OFF` prohibits software paging of device memory. A value of 0 means that *OptiX* is allowed to activate paging if necessary, 1 means that paging is always off. Note that currently paging cannot be disabled once it has been activated.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE - Can be returned if **size** does not match the proper size of the attribute, or if **p** is NULL.

HISTORY

rtContextSetAttribute was introduced in *OptiX 2.5*.

SEE ALSO

rtContextGetAttribute

1.1.25 `rtContextSetD3D9Device`

NAME

`rtContextSetD3D9Device` - Binds a D3D9 device to a context and enables interop

SYNOPSIS

```
#include <optix_d3d9_interop.h>

RTresult rtContextSetD3D9Device(RTcontext context,
                                IDirect3DDevice9* device);
```

PARAMETERS

`context`

The context to bind the device with.

`device`

The D3D9 device to be used for interop with the associated context.

DESCRIPTION

`rtContextSetD3D9Device` binds `device` to `context` and enables D3D9 interop capabilities in `context`. This function must be executed once for `context` before any call to `rtBufferCreateFromD3D9Resource` or `rtTextureSamplerCreateFromD3D9Resource` can take place. A context can only be bound to one device. Once `device` is bound to `context`, the binding is immutable and remains upon destruction of `context`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetD3D9Device` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D9Resource *rtTextureSamplerCreateFromD3D9Resource*

1.1.26 rtContextSetD3D10Device

NAME

rtContextSetD3D10Device - Binds a D3D10 device to a context and enables interop

SYNOPSIS

```
#include <optix_d3d10_interop.h>

RTresult rtContextSetD3D10Device(RTcontext context,
                                 IDirect3DDevice10* device);
```

PARAMETERS

context

The context to bind the device with.

device

The D3D10 device to be used for interop with the associated context.

DESCRIPTION

rtContextSetD3D10Device binds **device** to **context** and enables D3D10 interop capabilities in **context**. This function must be executed once for **context** before any call to **rtBufferCreateFromD3D10Resource** or **rtTextureSamplerCreateFromD3D10Resource** can take place. A context can only be bound to one device. Once **device** is bound to **context**, the binding is immutable and remains upon destruction of **context**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtContextSetD3D10Device was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D10Resource *rtTextureSamplerCreateFromD3D10Resource*

1.1.27 `rtContextSetD3D11Device`

NAME

`rtContextSetD3D11Device` - Binds a D3D11 device to a context and enables interop

SYNOPSIS

```
#include <optix_d3d11_interop.h>

RTresult rtContextSetD3D11Device(RTcontext context,
                                 IDirect3DDevice11* device);
```

PARAMETERS

`context`

The context to bind the device with.

`device`

The D3D11 device to be used for interop with the associated context.

DESCRIPTION

`rtContextSetD3D11Device` binds `device` to `context` and enables D3D11 interop capabilities in `context`. This function must be executed once for `context` before any call to `rtBufferCreateFromD3D11Resource` or `rtTextureSamplerCreateFromD3D11Resource` can take place. A context can only be bound to one device. Once `device` is bound to `context`, the binding is immutable and remains upon destruction of `context`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetD3D11Device` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D11Resource *rtTextureSamplerCreateFromD3D11Resource*

1.1.28 `rtContextSetDevices`

NAME

`rtContextSetDevices` - Specify a list of hardware devices to be used by the kernel.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetDevices(RTcontext context,
                             unsigned int count,
                             const int* devices)
```

PARAMETERS

context

The context to which the hardware list is applied.

count

The number of devices in the list.

devices

The list of devices.

DESCRIPTION

`rtContextSetDevices` specifies a list of hardware devices to be used during execution of the subsequent trace kernels.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_NO_DEVICE`

`RT_ERROR_INVALID_DEVICE`

HISTORY

`rtContextSetDevices` was introduced in *OptiX* 1.0.

SEE ALSO *rtContextGetDevices*, *rtContextGetDeviceCount*

1.1.29 `rtContextSetEntryPointCount`

NAME

`rtContextSetEntryPointCount` - Set the number of entry points for a given context.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextSetEntryPointCount(RTcontext context,  
                                     unsigned int num_entry_points)
```

PARAMETERS

`context`

The context to be modified.

`num_entry_points`

The number of entry points to use.

DESCRIPTION

`rtContextSetEntryPointCount` sets the number of entry points associated with the given context to `num_entry_points`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetEntryPointCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetEntryPointCount

1.1.30 `rtContextSetExceptionEnabled`

NAME

`rtContextSetExceptionEnabled` - Enable or disable an exception.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextSetExceptionEnabled(RTcontext context,
                                           RException exception,
                                           int enabled)
```

PARAMETERS

`context`

The context for which the exception is to be enabled or disabled.

`exception`

The exception which is to be enabled or disabled.

`enabled`

Nonzero to enable the exception, 0 to disable the exception.

DESCRIPTION

`rtContextSetExceptionEnabled` is used to enable or disable specific exceptions. If an exception is enabled, the exception condition is checked for at runtime, and the exception program is invoked if the condition is met. The exception program can query the type of the caught exception by calling `rtGetExceptionCode`. `exception` may take one of the following values:

```
RT_EXCEPTION_INDEX_OUT_OF_BOUNDS
RT_EXCEPTION_STACK_OVERFLOW
RT_EXCEPTION_BUFFER_INDEX_OUT_OF_BOUNDS
RT_EXCEPTION_INVALID_RAY
RT_EXCEPTION_INTERNAL_ERROR
RT_EXCEPTION_USER
RT_EXCEPTION_ALL
```

`RT_EXCEPTION_INDEX_OUT_OF_BOUNDS` checks that `rtIntersectChild` and `rtReportIntersection` are called with a valid index.

`RT_EXCEPTION_STACK_OVERFLOW` checks the runtime stack against overflow. The most common cause for an overflow is a too deep `rtTrace` recursion tree.

`RT_EXCEPTION_BUFFER_INDEX_OUT_OF_BOUNDS` checks every read and write access to `rtBuffer` objects to be within valid bounds.

`RT_EXCEPTION_INVALID_RAY` checks the each ray's origin and direction values against NaNs and infinity values.

`RT_EXCEPTION_INTERNAL_ERROR` indicates an unexpected internal error in the runtime.

`RT_EXCEPTION_USER` is used to enable or disable all user-defined exceptions. The reserved range of exception codes for user-defined exceptions starts at `RT_EXCEPTION_USER` (0x400) and ends at 0xFFFF. See `rtThrow` for more information.

`RT_EXCEPTION_ALL` is a placeholder value which can be used to enable or disable all possible exceptions with a single call to `rtContextSetExceptionEnabled`.

By default, `RT_EXCEPTION_STACK_OVERFLOW` is enabled and all other exceptions are disabled.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetExceptionEnabled` was introduced in *OptiX* 1.1.

SEE ALSO

rtContextGetExceptionEnabled, *rtContextSetExceptionProgram*, *rtContextGetExceptionProgram*, *rtGetExceptionCode*, *rtThrow*, *rtPrintExceptionDetails*

1.1.31 `rtContextSetExceptionProgram`

NAME

`rtContextSetExceptionProgram` - Specifies the exception program for a given context entry point.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtContextSetExceptionProgram(RTcontext context,  
                                     unsigned int entry_point_index,  
                                     RTprogram program)
```

PARAMETERS

`context`

The context node to which the exception program will be added.

`entry_point_index`

The entry point the program will be associated with.

`program`

The exception program.

DESCRIPTION

`rtContextSetExceptionProgram` sets `context`'s exception program at entry point `entry_point_index`. `RT_ERROR_INVALID_VALUE` is returned if `entry_point_index` is outside of the range `[0, rtContextGetEntryPointCount()-1]`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtContextSetExceptionProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetEntryPointCount, rtContextGetExceptionProgram, rtContextSetExceptionEnabled, rtContextGetExceptionEnabled, rtGetExceptionCode, rtThrow, rtPrintExceptionDetails

1.1.32 `rtContextSetMissProgram`

NAME

`rtContextSetMissProgram` - Specifies the miss program for a given context ray type.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetMissProgram(RTcontext context,
                                unsigned int ray_type_index,
                                RTprogram program)
```

PARAMETERS

`context`

The context node to which the miss program will be added.

`ray_type_index`

The ray type the program will be associated with.

`program`

The miss program.

DESCRIPTION

`rtContextSetMissProgram` sets `context`'s miss program associated with ray type `ray_type_index`. `RT_ERROR_INVALID_VALUE` is returned if `ray_type_index` is outside of the range `[0, rtContextGetRayTypeCount()-1]`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtContextSetMissProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetRayTypeCount, *rtContextGetMissProgram*

1.1.33 `rtContextSetPrintBufferSize`

NAME

`rtContextSetPrintBufferSize` - Set the size of the print buffer.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextSetPrintBufferSize(RTcontext context,  
                                           RTsize buffer_size_bytes)
```

PARAMETERS

`context`

The context for which to set the print buffer size.

`buffer_size_bytes`

The print buffer size in bytes.

DESCRIPTION

`rtContextSetPrintBufferSize` is used to set the buffer size available to hold data generated by `rtPrintf`. The default size is 65536 bytes.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetPrintBufferSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextSetPrintEnabled*, *rtContextGetPrintEnabled*, *rtContextGetPrintBufferSize*, *rtContextSetPrintLaunchIndex*, *rtContextGetPrintLaunchIndex*

1.1.34 `rtContextSetPrintEnabled`

NAME

`rtContextSetPrintEnabled` - Enable or disable text printing from programs.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextSetPrintEnabled(RTcontext context,  
                                        int enabled)
```

PARAMETERS

`context`

The context for which printing is to be enabled or disabled.

`enabled`

Setting this parameter to a nonzero value enables printing, 0 disables printing.

DESCRIPTION

`rtContextSetPrintEnabled` is used to control whether text printing in programs through `rtPrintf` is currently enabled for this context.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetPrintEnabled` was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextGetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextGetPrintBufferSize*, *rtContextSetPrintLaunchIndex*, *rtContextGetPrintLaunchIndex*

1.1.35 `rtContextSetPrintLaunchIndex`

NAME

`rtContextSetPrintLaunchIndex` - Sets the active launch index to limit text output.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult RTAPI rtContextSetPrintLaunchIndex(RTcontext context,
                                             int x,
                                             int y,
                                             int z)
```

PARAMETERS

`context`

The context for which to set the print launch index.

`x`

The launch index in the x dimension to which to limit the output of `rtPrintf` invocations. If set to -1, output is generated for all launch indices in the x dimension.

`y`

The launch index in the y dimension to which to limit the output of `rtPrintf` invocations. If set to -1, output is generated for all launch indices in the y dimension.

`z`

The launch index in the z dimension to which to limit the output of `rtPrintf` invocations. If set to -1, output is generated for all launch indices in the z dimension.

DESCRIPTION

`rtContextSetPrintLaunchIndex` is used to control for which launch indices `rtPrintf` generates output. The initial value of (x,y,z) is (-1,-1,-1), which generates output for all indices.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetPrintLaunchIndex` was introduced in *OptiX* 1.0.

SEE ALSO

rtPrintf, *rtContextGetPrintEnabled*, *rtContextSetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextGetPrintBufferSize*, *rtContextGetPrintLaunchIndex*

1.1.36 `rtContextSetRayGenerationProgram`

NAME

`rtContextSetRayGenerationProgram` - Specifies the ray generation program for a given context entry point.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetRayGenerationProgram(RTcontext context,
                                         unsigned int entry_point_index,
                                         RTprogram program)
```

PARAMETERS

`context`

The context node to which the exception program will be added.

`entry_point_index`

The entry point the program will be associated with.

`program`

The ray generation program.

DESCRIPTION

`rtContextSetRayGenerationProgram` sets `context`'s ray generation program at entry point `entry_point_index`. `RT_ERROR_INVALID_VALUE` is returned if `entry_point_index` is outside of the range `[0, rtContextGetEntryPointCount()-1]`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtContextSetRayGenerationProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetEntryPointCount, rtContextGetRayGenerationProgram

1.1.37 `rtContextSetRayTypeCount`

NAME

`rtContextSetRayTypeCount` - Sets the number of ray types for a given context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetRayTypeCount(RTcontext context,
                                   unsigned int num_ray_types)
```

PARAMETERS

context

The context node.

num_ray_types

The number of ray types to be used.

DESCRIPTION

`rtContextSetRayTypeCount` Sets the number of ray types associated with the given context.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetRayTypeCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetRayTypeCount

1.1.38 `rtContextSetStackSize`

NAME

`rtContextSetStackSize` - Set the stack size for a given context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetStackSize(RTcontext context,
                               RTsize stack_size_bytes)
```

PARAMETERS

`context`

The context node to be modified.

`stack_size_bytes`

The desired stack size in bytes.

DESCRIPTION

`rtContextSetStackSize` sets the stack size for the given context to `stack_size_bytes` bytes. Returns `RT_ERROR_INVALID_VALUE` if context is not valid.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetStackSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetStackSize

1.1.39 `rtContextSetTimeoutCallback`

NAME

`rtContextSetTimeoutCallback` - Set an application-side timeout callback function.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextSetTimeoutCallback(RTcontext context,
                                     RTtimeoutcallback callback,
                                     double min_polling_seconds)
```

PARAMETERS

`context`

The context node to be modified.

`callback`

The function to be called.

`min_polling_seconds`

The timeout interval after which the function is called.

DESCRIPTION

`rtContextSetTimeoutCallback` sets an application-side callback function `callback` and a time interval `min_polling_seconds` in seconds. Long-running *OptiX* API calls such as `rtContextCompile` and `rtContextLaunch` call the callback function about every `min_polling_seconds` seconds. If the callback function returns true, the API call tries to abort, leaving the context in a clean but unfinished state. Output buffers are left in an unpredictable state. In case an *OptiX* API call is terminated by a callback function, it returns `RT_TIMEOUT_CALLBACK`.

`RTtimeoutcallback` is defined as `int (*RTtimeoutcallback)(void)`.

To unregister a callback function, `callback` needs to be set to `NULL` and `min_polling_seconds` to 0.

Returns `RT_ERROR_INVALID_VALUE` if `context` is not valid, if `min_polling_seconds` is negative, if `callback` is `NULL` but `min_polling_seconds` is not 0, or if `callback` is not `NULL` but `min_polling_seconds` is 0.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtContextSetTimeoutCallback` was introduced in *OptiX 2.5*.

SEE ALSO

rtContextContextCompile, *rtContextLaunch*

1.1.40 `rtContextLaunch`

NAME

rtContextLaunch - Executes the computation kernel for a given context.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextLaunch1D(RTcontext context,
                          unsigned int entry_point_index,
                          RTsize image_width)

RTresult rtContextLaunch2D(RTcontext context,
                          unsigned int entry_point_index,
                          RTsize image_width,
                          RTsize image_height)

RTresult rtContextLaunch3D(RTcontext context,
                          unsigned int entry_point_index,
                          RTsize image_width,
                          RTsize image_height,
                          RTsize image_depth)
```

PARAMETERS

context

The context to be executed.

entry_point_index

The initial entry point into kernel.

image_width, image_height, image_depth

Specifies the size of the computation grid.

DESCRIPTION

rtContextLaunch executes the computation kernel associated with the given context. If the context has not yet been compiled, or if the context has been modified since the last compile, **rtContextLaunch** will recompile the kernel internally. Acceleration structures of the context which are marked dirty will be updated and their dirty flags will be cleared. Similarly, validation will occur if necessary. The ray generation program specified by **entry_point_index** will be invoked once for every element (pixel or voxel) of the computation grid specified by **image_width**, **image_height**, and **image_depth**.

`RT_ERROR_INVALID_SOURCE` is returned if t

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_INVALID_SOURCE

RT_ERROR_LAUNCH_FAILED

HISTORY

`rtContextLaunch` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextIsRunningState, *rtContextCompile*, *rtContextValidate*

1.1.41 `rtContextValidate`

NAME

`rtContextValidate` - Checks the given context for valid internal state.

SYNOPSIS

```
#include <optix.h>

RTresult rtContextValidate(RTcontext context)
```

PARAMETERS

`context`

The context to be validated.

DESCRIPTION

`rtContextValidate` checks the the given context and all of its associated *OptiX* objects for a valid state. These checks include tests for presence of necessary programs (eg. an intersection program for a geometry node), invalid internal state such as NULL children in graph nodes, and presence of variables required by all specified programs. `rtContextGetErrorString` can be used to retrieve a description of a validation failure.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_INVALID_SOURCE
```

HISTORY

`rtContextValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextGetErrorString

1.2 Geometry Group

NAME

Geometry Group

DESCRIPTION

This section describes the API functions for creation and handling of GeometryGroup objects.

`rtGeometryGroupCreate`

`rtGeometryGroupDestroy`

`rtGeometryGroupGetAcceleration`

`rtGeometryGroupGetChildCount`

`rtGeometryGroupGetChild`

`rtGeometryGroupGetContext`

`rtGeometryGroupSetAcceleration`

`rtGeometryGroupSetChildCount`

`rtGeometryGroupSetChild`

`rtGeometryGroupValidate`

HISTORY

GeometryGroup objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.2.1 `rtGeometryGroupCreate`

NAME

`rtGeometryGroupCreate` - Creates a new geometry group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGroupCreate(RTcontext context,
                               RTgeometrygroup* geometrygroup)
```

PARAMETERS

`context`

Specifies a context within which to create a new geometry group.

`geometrygroup`

Returns a newly created geometry group.

DESCRIPTION

`rtGeometryGroupCreate` creates a new geometry group within a context. `context` specifies the target context, and should be a value returned by `rtContextCreate`. After the call, `*geometrygroup` shall be set to the handle of a newly created group within `context`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupCreate` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupDestroy, *rtContextCreate*

1.2.2 `rtGeometryGroupDestroy`

NAME

`rtGeometryGroupDestroy` - Destroys a geometry group node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupDestroy(RTgeometrygroup geometrygroup)
```

PARAMETERS

`geometrygroup`

Handle of the geometry group node to destroy

DESCRIPTION

`rtGeometryGroupDestroy` removes `geometrygroup` from its context and deletes it. `geometrygroup` should be a value returned by `rtGeometryGroupCreate`. No child graph nodes are destroyed. After the call, `geometrygroup` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupCreate

1.2.3 `rtGeometryGroupGetAcceleration`

NAME

`rtGeometryGroupGetAcceleration` - Returns the acceleration structure attached to a geometry group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupGetAcceleration(RTgeometrygroup geometrygroup,  
                                        RTacceleration* acceleration)
```

PARAMETERS

`geometrygroup`

The geometry group handle.

`acceleration`

The returned acceleration structure object.

DESCRIPTION

`rtGeometryGroupGetAcceleration` returns the acceleration structure attached to a geometry group using **`rtGeometryGroupSetAcceleration`**. If no acceleration structure has previously been set, **`*acceleration`** is not written to, and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupGetAcceleration` was introduced in *OptiX* 1.0.

SEE ALSO

`rtGeometryGroupSetAcceleration`, *`rtAccelerationCreate`*

1.2.4 `rtGeometryGroupGetChildCount`

NAME

`rtGeometryGroupGetChildCount` - Returns the number of child slots for a group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupGetChildCount(RTgeometrygroup geometrygroup,  
                                       unsigned int* count)
```

PARAMETERS

`geometrygroup`

The parent geometry group handle.

`count`

Returned number of child slots.

DESCRIPTION

`rtGeometryGroupGetChildCount` returns the number of child slots allocated using `rtGeometryGroupSetChildCount`. This includes empty slots which may not yet have actual children assigned by `rtGeometryGroupSetChild`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupGetChildCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupSetChild, *rtGeometryGroupGetChild*, *rtGeometryGroupSetChildCount*, *rtGeometryGroupGetChildType*

1.2.5 rtGeometryGroupGetChild

NAME

rtGeometryGroupGetChild - Returns a child node of a geometry group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGroupGetChild(RTgeometrygroup geometrygroup,
                                  unsigned int index,
                                  RTgeometryinstance* geometryinstance)
```

PARAMETERS

geometrygroup

The parent geometry group handle.

index

The index of the child slot to query.

geometryinstance

The returned child geometry instance.

DESCRIPTION

rtGeometryGroupGetChild returns the child geometry instance at slot **index** of the parent **geometrygroup**. If no child has been assigned to the given slot, ***child** is not written to and **RT_ERROR_INVALID_VALUE** is returned.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtGeometryGroupGetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupSetChild, *rtGeometryGroupSetChildCount*, *rtGeometryGroupGetChildCount*, *rtGeometryGroupGetChildType*

1.2.6 `rtGeometryGroupGetContext`

NAME

`rtGeometryGroupGetContext` - Returns the context associated with a geometry group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupGetContext(RTgeometrygroup geometrygroup,  
                                   RTcontext* context)
```

PARAMETERS

geometrygroup

Specifies the geometry group to query.

context

Returns the context associated with the geometry group.

DESCRIPTION

`rtGeometryGroupGetContext` queries a geometry group for its associated context. **geometrygroup** specifies the geometry group to query, and must be a value returned by `rtGeometryGroupCreate`. After the call, ***context** shall be set to the context associated with **geometrygroup**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate, *rtGeometryGroupCreate*

1.2.7 `rtGeometryGroupSetAcceleration`

NAME

`rtGeometryGroupSetAcceleration` - Set the acceleration structure for a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGroupSetAcceleration(RTgeometrygroup geometrygroup,
                                        RTacceleration acceleration)
```

PARAMETERS

`geometrygroup`

The geometry group handle.

`acceleration`

The acceleration structure to attach to the geometry group.

DESCRIPTION

`rtGeometryGroupSetAcceleration` attaches an acceleration structure to a geometry group. The acceleration structure must have been previously created using `rtAccelerationCreate`. Every geometry group is required to have an acceleration structure assigned in order to pass validation. The acceleration structure will be built over the primitives contained in all children of the geometry group. This enables a single acceleration structure to be built over primitives of multiple geometry instances. Note that it is legal to attach a single `RTacceleration` object to multiple geometry groups, as long as the underlying geometry of all children is the same. This corresponds to attaching an acceleration structure to multiple groups at higher graph levels using `rtGroupSetAcceleration`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGeometryGroupSetAcceleration` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupGetAcceleration, rtAccelerationCreate, rtGroupSetAcceleration

1.2.8 `rtGeometryGroupSetChildCount`

NAME

`rtGeometryGroupSetChildCount` - Sets the number of child nodes to be attached to the group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupSetChildCount(RTgeometrygroup geometrygroup,  
                                       unsigned int count)
```

PARAMETERS

`geometrygroup`

The parent geometry group handle.

`count`

Number of child slots to allocate for the geometry group.

DESCRIPTION

`rtGeometryGroupSetChildCount` specifies the number of child slots in this geometry group. Potentially existing links to children at indices greater than `count-1` are removed. If the call increases the number of slots, the newly created slots are empty and need to be filled using `rtGeometryGroupSetChild` before validation.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupSetChildCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupGetChild, *rtGeometryGroupGetChildCount*, *rtGeometryGroupGetChildType*, *rtGeometryGroupSetChild*

1.2.9 `rtGeometryGroupSetChild`

NAME

`rtGeometryGroupSetChild` - Attaches a child node to a geometry group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGroupSetChild(RTgroup geometrygroup,
                                  unsigned int index,
                                  RTgeometryinstance geometryinstance)
```

PARAMETERS

`geometrygroup`

The parent geometry group handle.

`index`

The index in the parent's child slot array.

`geometryinstance`

The child node to be attached.

DESCRIPTION

`rtGeometryGroupSetChild` attaches a new child node `geometryinstance` to the parent node `geometrygroup`. `index` specifies the number of the slot where the child node gets attached. The index value must be lower than the number previously set by `rtGeometryGroupSetChildCount`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupSetChild` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupSetChildCount, *rtGeometryGroupGetChildCount*, *rtGeometryGroupGetChild*, *rtGeometryGroupGetChildType*

1.2.10 `rtGeometryGroupValidate`

NAME

`rtGeometryGroupValidate` - Validates the state of the geometry group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGroupValidate(RTgeometrygroup geometrygroup)
```

PARAMETERS

`geometrygroup`

Specifies the geometry group to be validated.

DESCRIPTION

`rtGeometryGroupValidate` checks `geometrygroup` for completeness. If `geometrygroup` or any of the objects attached to `geometrygroup` are not valid, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGroupValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGroupCreate

1.3 Group Node

NAME

Group Node

DESCRIPTION

This section describes the API functions for creation and handling of Group nodes.

rtGroupCreate

rtGroupDestroy

rtGroupGetAcceleration

rtGroupGetChildCount

rtGroupGetChild

rtGroupGetChildType

rtGroupGetContext

rtGroupSetAcceleration

rtGroupSetChildCount

rtGroupSetChild

rtGroupValidate

HISTORY

Group nodes were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.3.1 rtGroupCreate

NAME

rtGroupCreate - Creates a new group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupCreate(RTcontext context,
                      RTgroup* group)
```

PARAMETERS

context

Specifies a context within which to create a new group.

group

Returns a newly created group.

DESCRIPTION

rtGroupCreate creates a new group within a context. **context** specifies the target context, and should be a value returned by **rtContextCreate**. After the call, ***group** shall be set to the handle of a newly created group within **context**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtGroupCreate was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupDestroy, *rtContextCreate*

1.3.2 `rtGroupDestroy`

NAME

`rtGroupDestroy` - Destroys a group node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGroupDestroy(RTgroup group)
```

PARAMETERS

`group`

Handle of the group node to destroy.

DESCRIPTION

`rtGroupDestroy` removes `group` from its context and deletes it. `group` should be a value returned by `rtGroupCreate`. No child graph nodes are destroyed. After the call, `group` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupCreate

1.3.3 rtGroupGetAcceleration

NAME

rtGroupGetAcceleration - Returns the acceleration structure attached to a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupGetAcceleration(RTgroup group,
                                RTacceleration* acceleration)
```

PARAMETERS

group

The group handle.

acceleration

The returned acceleration structure object.

DESCRIPTION

rtGroupGetAcceleration returns the acceleration structure attached to a group using **rtGroupSetAcceleration**. If no acceleration structure has previously been set, ***acceleration** is not written to, and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

rtGroupGetAcceleration was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupSetAcceleration, *rtAccelerationCreate*

1.3.4 `rtGroupGetChildCount`

NAME

`rtGroupGetChildCount` - Returns the number of child slots for a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupGetChildCount(RTgroup group,
                              unsigned int* count)
```

PARAMETERS

group

The parent group handle.

count

Returned number of child slots.

DESCRIPTION

`rtGroupGetChildCount` returns the number of child slots allocated using `rtGroupSetChildCount`. This includes empty slots which may not yet have actual children assigned by `rtGroupSetChild`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupGetChildCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupSetChild, *rtGroupGetChild*, *rtGroupSetChildCount*, *rtGroupGetChildType*

1.3.5 rtGroupGetChild

NAME

rtGroupGetChild - Returns a child node of a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupGetChild(RTgroup group,
                        unsigned int index,
                        RObject* child)
```

PARAMETERS

group

The parent group handle.

index

The index of the child slot to query.

child

The returned child object.

DESCRIPTION

rtGroupGetChild returns the child object at slot **index** of the parent **group**. If no child has been assigned to the given slot, ***child** is not written to and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

rtGroupGetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupSetChild, *rtGroupSetChildCount*, *rtGroupGetChildCount*, *rtGroupGetChildType*

1.3.6 `rtGroupGetChildType`

NAME

`rtGroupGetChildType` - Get the type of a group child.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupGetChildType(RTgroup group,
                             unsigned int index,
                             RObjecttype* type)
```

PARAMETERS

group

The parent group handle.

index

The index of the child slot to query.

type

The returned child type.

DESCRIPTION

`rtGroupGetChildType` returns the type of the group child at slot **index**. If no child is associated with the given index, **type** is not written to and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupGetChildType` was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupSetChild, *rtGroupGetChild*, *rtGroupSetChildCount*, *rtGroupGetChildCount*

1.3.7 `rtGroupGetContext`

NAME

`rtGroupGetContext` - Returns the context associated with a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupGetContext(RTgroup group,
                           RTcontext* context)
```

PARAMETERS

group

Specifies the group to query.

context

Returns the context associated with the group.

DESCRIPTION

`rtGroupGetContext` queries a group for its associated context. **group** specifies the group to query, and must be a value returned by `rtGroupCreate`. After the call, ***context** shall be set to the context associated with **group**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate, *rtGroupCreate*

1.3.8 `rtGroupSetAcceleration`

NAME

`rtGroupSetAcceleration` - Set the acceleration structure for a group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupSetAcceleration(RTgroup group,
                                RTacceleration acceleration)
```

PARAMETERS

group

The group handle.

acceleration

The acceleration structure to attach to the group.

DESCRIPTION

`rtGroupSetAcceleration` attaches an acceleration structure to a group. The acceleration structure must have been previously created using **`rtAccelerationCreate`**. Every group is required to have an acceleration structure assigned in order to pass validation. The acceleration structure will be built over the children of the group. For example, if an acceleration structure is attached to a group that has a selector, a geometry group, and a transform child, the acceleration structure will be built over the bounding volumes of these three objects.

Note that it is legal to attach a single `RTacceleration` object to multiple groups, as long as the underlying bounds of the children are the same. For example, if another group has three children which are known to have the same bounding volumes as the ones in the example above, the two groups can share an acceleration structure, thus saving build time. This is true even if the details of the children, such as the actual type of a node or its geometry content, differ from the first set of group children. All that is required is for a child node at a given index to have the same bounds as the other group's child node at the same index.

Sharing an acceleration structure this way corresponds to attaching an acceleration structure to multiple geometry groups at lower graph levels using **`rtGeometryGroupSetAcceleration`**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

rtGroupSetAcceleration was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupGetAcceleration, *rtAccelerationCreate*, *rtGeometryGroupSetAcceleration*

1.3.9 `rtGroupSetChildCount`

NAME

`rtGroupSetChildCount` - Sets the number of child nodes to be attached to the group.

SYNOPSIS

```
#include <optix.h>

RTresult rtGroupSetChildCount(RTgroup group,
                              unsigned int count)
```

PARAMETERS

`group`

The parent group handle.

`count`

Number of child slots to allocate for the group.

DESCRIPTION

`rtGroupSetChildCount` specifies the number of child slots in this group. Potentially existing links to children at indices greater than `count-1` are removed. If the call increases the number of slots, the newly created slots are empty and need to be filled using `rtGroupSetChild` before validation.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupSetChildCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupGetChild, *rtGroupGetChildCount*, *rtGroupGetChildType*, *rtGroupSetChild*

1.3.10 rtGroupSetChild

NAME

rtGroupSetChild - Attaches a child node to a group.

SYNOPSIS

```
#include <optix.h>

RtResult rtGroupSetChild(Rtgroup group,
                        unsigned int index,
                        Rtobject child)
```

PARAMETERS

group

The parent group handle.

index

The index in the parent's child slot array.

child

The child node to be attached. Can be of type **{Rtgroup, Rtselector, Rtgeometrygroup, Rttransform}**.

DESCRIPTION

Attaches a new child node **child** to the parent node **group**. **index** specifies the number of the slot where the child node gets attached. A sufficient number of slots must be allocated using **rtGroupSetChildCount**. Legal child node types are **Rtgroup**, **Rtselector**, **Rtgeometrygroup**, and **Rttransform**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtGroupSetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupSetChildCount, rtGroupGetChildCount, rtGroupGetChild, rtGroupGetChildType

1.3.11 `rtGroupValidate`

NAME

`rtGroupValidate` - Verifies the state of the group.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGroupValidate(RTgroup group)
```

PARAMETERS

`group`

Specifies the group to be validated.

DESCRIPTION

`rtGroupValidate` checks `group` for completeness. If `group` or any of the objects attached to `group` are not valid, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGroupValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtGroupCreate

1.4 Selector Node

NAME

Selector Node

DESCRIPTION

This section describes the API functions for creation and handling of Selector nodes. Selector nodes are used to dynamically switch between model sub-trees in the scene during ray traversal. This can, e.g., be used to implement level-of-detail rendering.

rtSelectorCreate

rtSelectorDeclareVariable

rtSelectorDestroy

rtSelectorGetChildCount

rtSelectorGetChild

rtSelectorGetChildType

rtSelectorGetContext

rtSelectorGetVariableCount

rtSelectorGetVariable

rtSelectorGetVisitProgram

rtSelectorQueryVariable

rtSelectorRemoveVariable

rtSelectorSetChildCount

rtSelectorSetChild

rtSelectorSetVisitProgram

rtSelectorValidate

HISTORY

Selector nodes were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Transform Node, Acceleration Structure, Geometry Instance, *Geometry*, *Material*, *Program*, *Buffer*, Texture Sampler, *Variables*, Context-Free Functions

1.4.1 rtSelectorCreate

NAME

rtSelectorCreate - creates a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorCreate(RTcontext context,
                          RTselector* selector)
```

PARAMETERS

context

Specifies the rendering context of the Selector node.

selector

New Selector node handle.

DESCRIPTION

Creates a new Selector node within the given context. After calling **rtSelectorCreate()** the new node is in a "raw" state. For the node to be functional, a visit program has to be assigned using **rtSelectorSetVisitProgram()**. Furthermore, a number of (zero or more) children can be attached by using **rtSelectorSetChildCount()** and **rtSelectorSetChild()**.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorCreate was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorDestroy, *rtSelectorValidate*, *rtSelectorGetContext*, *rtSelectorSetVisitProgram*, *rtSelectorSetChildCount*, *rtSelectorSetChild*

1.4.2 rtSelectorDeclareVariable

NAME

`rtSelectorDeclareVariable` - declares a variable associated with a Selector node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtSelectorDeclareVariable(RTselector selector,  
                                   const char* name,  
                                   RTvariable* v)
```

PARAMETERS

selector

Selector node handle.

name

Variable identifier.

v

New variable handle.

DESCRIPTION

Declares a new variable identified by **name**, and associates it with the Selector node **selector**. The new variable handle is returned in **v**. After declaration, a variable does not have a type until its value is set by an `rtVariableSet{...}()` function. Once a variable type has been set, it cannot be changed, i.e., only `rtVariableSet{...}()` functions of the same type can be used to change the value of the variable.

RETURN VALUES

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_REDECLARED`

`RT_ERROR_ILLEGAL_SYMBOL`

HISTORY

`rtSelectorDeclareVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorQueryVariable, *rtSelectorRemoveVariable*, *rtSelectorGetVariableCount*, *rtSelectorGetVariable*, *rtVariableSet{...}*

1.4.3 rtSelectorDestroy

NAME

rtSelectorDestroy - Destroys a selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorDestroy(RTselector selector)
```

PARAMETERS

selector

Handle of the selector node to destroy

DESCRIPTION

rtSelectorDestroy removes **selector** from its context and deletes it. **selector** should be a value returned by **rtSelectorCreate**. Associated variables declared via **rtSelectorDeclareVariable** are destroyed, but no child graph nodes are destroyed. After the call, **selector** is no longer a valid handle.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

rtSelectorDestroy was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorCreate, *rtSelectorValidate*, *rtSelectorGetContext*

1.4.4 rtSelectorGetChildCount

NAME

rtSelectorGetChildCount - returns the number of child node slots of a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorGetChildCount(RTselector selector,
                                unsigned int* count)
```

PARAMETERS

selector

Selector node handle.

count

Number of child node slots reserved for **selector**.

DESCRIPTION

rtSelectorGetChildCount() returns in **count** the number of child node slots that have been previously reserved for the Selector node **selector** by **rtSelectorSetChildCount()**. The value of **count** does *not* reflect the actual number of child nodes that have so far been attached to the Selector node using **rtSelectorSetChild()**.

RETURN VALUES

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetChildCount was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetChildCount, *rtSelectorSetChild*, *rtSelectorGetChild*, *rtSelectorGetChildType*

1.4.5 rtSelectorGetChild

NAME

rtSelectorGetChild - returns a child node that is attached to a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorGetChild(RTselector selector,
                           unsigned int index,
                           RObject* child)
```

PARAMETERS

selector

Selector node handle.

index

Child node index.

child

Child node handle. Can be {**RTgroup**, **RTselector**, **RTgeometrygroup**, **RTtransform**}.

DESCRIPTION

rtSelectorGetChild() returns in **child** a handle of the child node currently attached to **selector** at slot **index**. The index value must be lower than the number previously set by **rtSelectorSetChildCount()**, thus it has to be in the range from **0** to **rtSelectorGetChildCount()-1**. The returned pointer is of generic type **RObject** and needs to be cast to the actual child type, which can be **RTgroup**, **RTselector**, **RTgeometrygroup**, or **RTtransform**. The actual type of **child** can be queried using **rtSelectorGetChildType()**;

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetChildCount, rtSelectorGetChildCount, rtSelectorSetChild, rtSelectorGetChildType

1.4.6 rtSelectorGetChildType

NAME

rtSelectorGetChildType - returns type information about a Selector child node

SYNOPSIS

```
#include <optix.h>

Rtresult rtSelectorGetChildType(Rtselector selector,
                                unsigned int index,
                                RObjecttype* type)
```

PARAMETERS

selector

Selector node handle.

index

Child node index.

type

Type of the child node.

DESCRIPTION

rtSelectorGetChildType() queries the type of the child node attached to **selector** at slot **index**. The index value has to be in the range from **0** to **rtSelectorGetChildCount()-1**. The returned type is one of:

```
RT_OBJECTTYPE_GROUP
RT_OBJECTTYPE_GEOMETRY_GROUP
RT_OBJECTTYPE_TRANSFORM
RT_OBJECTTYPE_SELECTOR
```

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtSelectorGetChildType` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetChildCount, *rtSelectorGetChildCount*, *rtSelectorSetChild*, *rtSelectorGetChild*

1.4.7 rtSelectorGetContext

NAME

rtSelectorGetContext - returns the context of a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorGetContext(RTselector selector,
                             RTcontext* context)
```

PARAMETERS

selector

Selector node handle.

context

The context, **selector** belongs to.

DESCRIPTION

rtSelectorGetContext() returns in **context** the rendering context in which the Selector node **selector** has been created.

RETURN VALUES

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetContext was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorCreate, *rtSelectorDestroy*, *rtSelectorValidate*

1.4.8 rtSelectorGetVariableCount

NAME

rtSelectorGetVariableCount - returns the number of variables attached to a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorGetVariableCount(RTselector selector,
                                    unsigned int* count)
```

PARAMETERS

selector

Selector node handle.

count

Number of variables associated with **selector**.

DESCRIPTION

rtSelectorGetVariableCount() returns in **count** the number of variables that are currently attached to the Selector node **selector**.

RETURN VALUES

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetVariableCount was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorDeclareVariable, *rtSelectorQueryVariable*, *rtSelectorRemoveVariable*, *rtSelectorGetVariable*

1.4.9 rtSelectorGetVariable

NAME

rtSelectorGetVariable - returns a variable associated with a Selector node

SYNOPSIS

```
#include <optix.h>

RtResult rtSelectorGetVariable(Rtselector selector,
                               unsigned int index,
                               Rtvariable* v)
```

PARAMETERS

selector

Selector node handle.

index

Variable index.

v

Variable handle.

DESCRIPTION

Returns in **v** a handle to the variable located at position **index** in the Selector's variable array. **index** is a sequential number depending on the order of variable declarations. The index has to be in the range from **0** to **rtSelectorGetVariableCount()-1**. The current value of a variable can be retrieved from its handle by using an appropriate **rtVariableGet{...}()** function matching the variable's type.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetVariable was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorDeclareVariable, *rtSelectorQueryVariable*, *rtSelectorRemoveVariable*, *rtSelectorGetVariableCount*,
rtVariableGet{...}

1.4.10 rtSelectorGetVisitProgram

NAME

rtSelectorGetVisitProgram - returns the currently assigned visit program

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorGetVisitProgram(RTselector selector,
                                   RTprogram* program)
```

PARAMETERS

selector

Selector node handle.

program

Current visit program assigned to **selector**.

DESCRIPTION

rtSelectorGetVisitProgram() returns in **program** a handle of the visit program currently bound to **selector**.

RETURN VALUES

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorGetVisitProgram was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetVisitProgram

1.4.11 rtSelectorQueryVariable

NAME

rtSelectorQueryVariable - returns a variable associated with a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorQueryVariable(RTselector selector,
                                const char* name,
                                RTvariable* v)
```

PARAMETERS

selector

Selector node handle.

name

Variable identifier.

v

Variable handle.

DESCRIPTION

Returns in **v** a handle to the variable identified by **name**, which is associated with the Selector node **selector**. The current value of a variable can be retrieved from its handle by using an appropriate **rtVariableGet{...}()** function matching the variable's type.

RETURN VALUES

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

rtSelectorQueryVariable was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorDeclareVariable, *rtSelectorRemoveVariable*, *rtSelectorGetVariableCount*, *rtSelectorGetVariable*, *rtVariableGet{...}*

1.4.12 rtSelectorRemoveVariable

NAME

rtSelectorRemoveVariable - removes a variable from a Selector node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtSelectorRemoveVariable(RTselector selector,  
                                  RTvariable v)
```

PARAMETERS

selector

Selector node handle.

v

Variable handle.

DESCRIPTION

rtSelectorRemoveVariable() removes the variable **v** from the Selector node **selector** and deletes it. The handle **v** must be considered invalid afterwards.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_VARIABLE_NOT_FOUND

HISTORY

rtSelectorRemoveVariable was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorDeclareVariable, *rtSelectorQueryVariable*, *rtSelectorGetVariableCount*, *rtSelectorGetVariable*

1.4.13 `rtSelectorSetChildCount`

NAME

`rtSelectorSetChildCount` - specifies the number of child nodes to be attached to a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorSetChildCount(RTselector selector,
                                unsigned int count)
```

PARAMETERS

`selector`

Selector node handle.

`count`

Number of child nodes to be attached to `selector`.

DESCRIPTION

`rtSelectorSetChildCount()` allocates a number of children slots, i.e., it pre-defines the *exact* number of child nodes the parent Selector node `selector` will have. Child nodes have to be attached to the Selector node using `rtSelectorSetChild()`. Empty slots will cause a validation error.

RETURN VALUES

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

`rtSelectorSetChildCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorValidate, *rtSelectorGetChildCount*, *rtSelectorSetChild*, *rtSelectorGetChild*, *rtSelectorGetChildType*

1.4.14 rtSelectorSetChild

NAME

rtSelectorSetChild - attaches a child node to a Selector node

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorSetChild(RTselector selector,
                           unsigned int index,
                           RObject child)
```

PARAMETERS

selector

Selector node handle.

index

Index of the parent slot the node **child** gets attached to.

child

Child node to be attached. Can be {**RTgroup**, **RTselector**, **RTgeometrygroup**, **RTtransform**}.

DESCRIPTION

Attaches a new child node **child** to the parent node **selector**. **index** specifies the number of the slot where the child node gets attached. The index value must be lower than the number previously set by **rtSelectorSetChildCount()**, thus it has to be in the range from **0** to **rtSelectorGetChildCount()-1**. Legal child node types are **RTgroup**, **RTselector**, **RTgeometrygroup**, and **RTtransform**.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtSelectorSetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetChildCount, rtSelectorGetChildCount, rtSelectorGetChild, rtSelectorGetChildType

1.4.15 `rtSelectorSetVisitProgram`

NAME

`rtSelectorSetVisitProgram` - assigns a visit program to a Selector node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtSelectorSetVisitProgram(RTselector selector,  
                                   RTprogram program)
```

PARAMETERS

`selector`

Selector node handle.

`program`

Program handle associated with a visit program.

DESCRIPTION

`rtSelectorSetVisitProgram()` specifies a visit program that is executed when the Selector node `selector` gets visited by a ray during traversal of the model graph. A visit program steers how traversal of the Selector's children is performed. It usually chooses only a single child to continue traversal, but is also allowed to process zero or multiple children. Programs can be created from PTX files using `rtProgramCreateFromPTXFile()`.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_TYPE_MISMATCH

HISTORY

`rtSelectorSetVisitProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorGetVisitProgram, rtProgramCreateFromPTXFile

1.4.16 `rtSelectorValidate`

NAME

`rtSelectorValidate` - checks a Selector node for internal consistency

SYNOPSIS

```
#include <optix.h>

RTresult rtSelectorValidate(RTselector selector)
```

PARAMETERS

`selector`

Selector root node of a model sub-tree to be validated.

DESCRIPTION

`rtSelectorValidate()` recursively checks consistency of the Selector node **selector** and its children, i.e., it tries to validate the whole model sub-tree with **selector** as root. For a Selector node to be valid, it must be assigned a visit program, and the number of its children must match the number specified by `rtSelectorSetChildCount()`.

RETURN VALUES

`RT_SUCCESS`
`RT_ERROR_INVALID_CONTEXT`
`RT_ERROR_INVALID_VALUE`
`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtSelectorValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorCreate, *rtSelectorDestroy*, *rtSelectorGetContext*, *rtSelectorSetVisitProgram*, *rtSelectorSetChildCount*, *rtSelectorSetChild*

1.5 Transform Node

NAME

Transform Node

DESCRIPTION

This section describes the API functions for creation and handling of Transform nodes. Transform nodes are used to perform affine geometrical transformations on sub-trees of the scene.

rtTransformCreate

rtTransformDestroy

rtTransformGetChild

rtTransformGetChildType

rtTransformGetContext

rtTransformGetMatrix

rtTransformSetChild

rtTransformSetMatrix

rtTransformValidate

HISTORY

Transform nodes were introduced in *OptiX* 1.0.

SEE ALSO

Context, *Geometry Group*, *Group Node*, *Selector Node*, *Acceleration Structure*, *Geometry Instance*, *Geometry*, *Material*, *Program*, *Buffer*, *Texture Sampler*, *Variables*, *Context-Free Functions*

1.5.1 `rtTransformCreate`

NAME

`rtTransformCreate` - creates a new Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformCreate(RTcontext context,
                          RTtransform* transform)
```

PARAMETERS

`context`

Specifies the rendering context of the Transform node.

`selector`

New Transform node handle.

DESCRIPTION

Creates a new Transform node within the given context. For the node to be functional, a child node has to be attached using `rtTransformSetChild()`. A transformation matrix can be associated with the transform node with `rtTransformSetMatrix()`.

RETURN VALUES

`RT_SUCCESS`
`RT_ERROR_INVALID_CONTEXT`
`RT_ERROR_INVALID_VALUE`
`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTransformCreate` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformDestroy, *rtTransformValidate*, *rtTransformGetContext*, *rtTransformSetMatrix*, *rtTransformGetMatrix*, *rtTransformSetChild*, *rtTransformGetChild*, *rtTransformGetChildTypeChild*

1.5.2 `rtTransformDestroy`

NAME

`rtTransformDestroy` - Destroys a transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformDestroy(RTtransform transform)
```

PARAMETERS

`transform`

Handle of the transform node to destroy

DESCRIPTION

`rtTransformDestroy` removes `transform` from its context and deletes it. `transform` should be a value returned by `rtTransformCreate`. No child graph nodes are destroyed. After the call, `transform` is no longer a valid handle.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

`rtTransformDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformValidate*, *rtTransformGetContext*

1.5.3 rtTransformGetChild

NAME

rtTransformGetChild - returns the child node that is attached to a Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformGetChild(RTtransform transform,
                             RObject* child)
```

PARAMETERS

transform

Transform node handle.

child

Child node handle. Can be {**RTgroup**, **RTselector**, **RTgeometrygroup**, **RTtransform**}.

DESCRIPTION

rtTransformGetChild() returns in **child** a handle of the child node currently attached to **transform**. The returned pointer is of generic type **RObject** and needs to be cast to the actual child type, which can be **RTgroup**, **RTselector**, **RTgeometrygroup**, or **RTtransform**. The actual type of **child** can be queried using **rtTransformGetChildType()**.

RETURN VALUES

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtTransformGetChild was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformSetChild, *rtTransformGetChildType*

1.5.4 `rtTransformGetChildType`

NAME

`rtTransformGetChildType` - returns type information about a Transform child node

SYNOPSIS

```
#include <optix.h>

Rtresult rtTransformGetChildType(Rttransform transform,
                                RObjecttype* type)
```

PARAMETERS

`transform`

Transform node handle.

`type`

Type of the child node.

DESCRIPTION

`rtTransformGetChildType()` queries the type of the child node attached to **selector**. The returned type is one of:

```
RT_OBJECTTYPE_GROUP
RT_OBJECTTYPE_GEOMETRY_GROUP
RT_OBJECTTYPE_TRANSFORM
RT_OBJECTTYPE_SELECTOR
```

RETURN VALUES

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

`rtTransformGetChildType` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformSetChild, rtTransformGetChild

1.5.5 `rtTransformGetContext`

NAME

`rtTransformGetContext` - returns the context of a Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformGetContext(RTtransform transform,
                              RTcontext* context)
```

PARAMETERS

`transform`

Transform node handle.

`context`

The context associated with `transform`.

DESCRIPTION

`rtTransformGetContext` queries a transform node for its associated context. `transform` specifies the transform node to query, and should be a value returned by `rtTransformCreate`. After the call, `*context` shall be set to the context associated with `transform`.

RETURN VALUES

`RT_SUCCESS`
`RT_ERROR_INVALID_CONTEXT`
`RT_ERROR_INVALID_VALUE`
`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTransformGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformDestroy*, *rtTransformValidate*

1.5.6 rtTransformGetMatrix

NAME

rtTransformGetMatrix - returns the affine matrix and its inverse associated with a Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformGetMatrix(RTtransform transform,
                              int transpose, float* matrix,
                              float* inverse_matrix)
```

PARAMETERS

transform

Transform node handle.

transpose

Flag indicating whether **matrix** and **inverse_matrix** should be transposed.

matrix

Affine matrix (4x4 float array).

inverse_matrix

Inverted form of **matrix**.

DESCRIPTION

rtTransformGetMatrix() returns in **matrix** the affine matrix that is currently used to perform a transformation of the geometry contained in the sub-tree with **transform** as root. The corresponding inverse matrix will be returned in **inverse_matrix**. One or both pointers are allowed to be NULL. If **transpose** is 0, matrices are returned in row-major format, i.e., matrix rows are contiguously laid out in memory. If **transpose** is non-zero, matrices are returned in column-major format. If non-NULL, matrix pointers must point to a float array of at least 16 elements.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTransformGetMatrix` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformSetMatrix

1.5.7 `rtTransformSetChild`

NAME

`rtTransformSetChild` - attaches a child node to a Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformSetChild(RTtransform transform,
                             RTOBJECT child)
```

PARAMETERS

`transform`

Transform node handle.

`child`

Child node to be attached. Can be {`RTgroup`, `RTselector`, `RTgeometrygroup`, `RTtransform`}.

DESCRIPTION

Attaches a child node `child` to the parent node `transform`. Legal child node types are `RTgroup`, `RTselector`, `RTgeometrygroup`, and `RTtransform`. A transform node must have exactly one child. If a transformation matrix has been attached to `transform` with `rtTransformSetMatrix()`, it is effective on the model sub-tree with `child` as root node.

RETURN VALUES

`RT_SUCCESS`
`RT_ERROR_INVALID_CONTEXT`
`RT_ERROR_INVALID_VALUE`
`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTransformSetChild` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformSetMatrix, *rtTransformGetChild*, *rtTransformGetChildType*

1.5.8 rtTransformSetMatrix

NAME

rtTransformSetMatrix - associates an affine transformation matrix with a Transform node

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformSetMatrix(RTtransform transform,
                              int transpose,
                              const float* matrix,
                              const float* inverse_matrix)
```

PARAMETERS

transform

Transform node handle.

transpose

Flag indicating whether **matrix** and **inverse_matrix** should be transposed.

matrix

Affine matrix (4x4 float array).

inverse_matrix

Inverted form of **matrix**.

DESCRIPTION

rtTransformSetMatrix() associates a 4x4 matrix with the Transform node **transform**. The provided transformation matrix results in a corresponding affine transformation of all geometry contained in the subtree with **transform** as root. At least one of the pointers **matrix** and **inverse_matrix** must be non-NULL. If exactly one pointer is valid, the other matrix will be computed. If both are valid, the matrices will be used as-is. If **transpose** is 0, source matrices are expected to be in row-major format, i.e., matrix rows are contiguously laid out in memory:

```
float matrix[4*4] = { a11,  a12,  a13,  a14,
                    a21,  a22,  a23,  a24,
                    a31,  a32,  a33,  a34,
                    a41,  a42,  a43,  a44 };
```

Here, the translational elements **a14**, **a24**, and **a34** are at the 4th, 8th, and 12th position the matrix array. If the supplied matrices are in column-major format, a non-0 **transpose** flag can be used to trigger an automatic transpose of the input matrices.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtTransformSetMatrix was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformGetMatrix

1.5.9 `rtTransformValidate`

NAME

`rtTransformValidate` - checks a Transform node for internal consistency

SYNOPSIS

```
#include <optix.h>

RTresult rtTransformValidate(RTtransform transform)
```

PARAMETERS

`transform`

Transform root node of a model sub-tree to be validated.

DESCRIPTION

`rtTransformValidate()` recursively checks consistency of the Transform node **transform** and its child, i.e., it tries to validate the whole model sub-tree with **transform** as root. For a Transform node to be valid, it must have a child node attached. It is, however, not required to explicitly set a transformation matrix. Without a specified transformation matrix, the identity matrix is applied.

RETURN VALUES

`RT_SUCCESS`
`RT_ERROR_INVALID_CONTEXT`
`RT_ERROR_INVALID_VALUE`
`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTransformValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformDestroy*, *rtTransformGetContext*, *rtTransformSetMatrix*, *rtTransformSetChild*

1.6 Acceleration Structure

NAME

Acceleration Structure

DESCRIPTION

This section describes the API functions for creation and handling of Acceleration structure objects.

rtAccelerationCreate

rtAccelerationDestroy

rtAccelerationGetBuilder

rtAccelerationGetContext

rtAccelerationGetData

rtAccelerationGetDataSize

rtAccelerationGetProperty

rtAccelerationGetTraverser

rtAccelerationIsDirty

rtAccelerationMarkDirty

rtAccelerationSetBuilder

rtAccelerationSetData

rtAccelerationSetProperty

rtAccelerationSetTraverser

rtAccelerationValidate

HISTORY

Acceleration structure objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Geometry Instance, Geometry, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.6.1 `rtAccelerationCreate`

NAME

`rtAccelerationCreate` - Creates a new acceleration structure.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationCreate(RTcontext context,  
                             RTacceleration* acceleration)
```

PARAMETERS

`context`

Specifies a context within which to create a new acceleration structure.

`acceleration`

Returns the newly created acceleration structure.

DESCRIPTION

`rtAccelerationCreate` creates a new ray tracing acceleration structure within a context. An acceleration structure is used by attaching it to a group or geometry group by calling `rtGroupSetAcceleration` or `rtGeometryGroupSetAcceleration`. Note that an acceleration structure can be shared by attaching it to multiple groups or geometry groups if the underlying geometric structures are the same, see `rtGroupSetAcceleration` and `rtGeometryGroupSetAcceleration` for more details. A newly created acceleration structure is initially in *dirty* state.

RETURN VALUES

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationCreate` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationDestroy, rtContextCreate, rtAccelerationMarkDirty, rtAccelerationIsDirty, rtGroupSetAcceleration, rtGeometryGroupSetAcceleration

1.6.2 `rtAccelerationDestroy`

NAME

`rtAccelerationDestroy` - Destroys an acceleration structure object

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationDestroy(RTacceleration acceleration)
```

PARAMETERS

`acceleration`

Handle of the acceleration structure to destroy

DESCRIPTION

`rtAccelerationDestroy` removes `acceleration` from its context and deletes it. `acceleration` should be a value returned by `rtAccelerationCreate`. After the call, `acceleration` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationCreate

1.6.3 `rtAccelerationGetBuilder`

NAME

`rtAccelerationGetBuilder` - Query the current builder from an acceleration structure.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationGetBuilder(RTacceleration acceleration,
                                  char** return_string)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`return_string`

Return string buffer.

DESCRIPTION

`rtAccelerationGetBuilder` returns the name of the builder currently used in the acceleration structure `acceleration`. If no builder has been set for `acceleration`, an empty string is returned. `return_string` will be set to point to the returned string. The memory `return_string` points to will be valid until the next API call that returns a string.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetBuilder` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationSetBuilder

1.6.4 `rtAccelerationGetContext`

NAME

`rtAccelerationGetContext` - Returns the context associated with an acceleration structure.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationGetContext(RTacceleration acceleration,  
                                 RTcontext* context)
```

PARAMETERS

acceleration

The acceleration structure handle.

context

Returns the context associated with the acceleration structure.

DESCRIPTION

`rtAccelerationGetContext` queries an acceleration structure for its associated context. The context handle is returned in the location pointed to by **context**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationCreate

1.6.5 `rtAccelerationGetData`

NAME

`rtAccelerationGetData` - Retrieves acceleration structure data.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationGetData(RTacceleration acceleration,
                              void* data)
```

PARAMETERS

acceleration

The acceleration structure handle.

data

Pointer to a memory region to be filled with the state of **acceleration**.

DESCRIPTION

`rtAccelerationGetData` retrieves the full state of the **acceleration** object, and copies it to the memory region pointed to by **data**. Sufficient memory must be available starting at that location to hold the entire state. To query the required memory size, `rtAccelerationGetDataSize` should be used.

The returned **data** from this call is valid input data for `rtAccelerationSetData`.

If **acceleration** is marked dirty, this call is invalid and will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetData` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationSetData, *rtAccelerationGetDataSize*

1.6.6 `rtAccelerationGetDataSize`

NAME

`rtAccelerationGetDataSize` - Returns the size of the data to be retrieved from an acceleration structure.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationGetDataSize(RTacceleration acceleration,
                                   RTsize* size)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`size`

The returned size of the data in bytes.

DESCRIPTION

`rtAccelerationGetDataSize` queries the size of the data that will be returned on a subsequent call to `rtAccelerationGetData`. The size in bytes will be written to the location pointed to by `size`. The returned value is guaranteed to be valid only if no other function using the handle `acceleration` is made before `rtAccelerationGetData`.

If `acceleration` is marked dirty, this call is invalid and will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetDataSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationGetData, *rtAccelerationSetData*

1.6.7 `rtAccelerationGetProperty`

NAME

`rtAccelerationGetProperty` - Queries an acceleration structure property.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationGetProperty(RTacceleration acceleration,  
                                  const char* name,  
                                  char** return_string)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`name`

The name of the property to be queried.

`return_string`

Return string buffer.

DESCRIPTION

`rtAccelerationGetProperty` returns the value of the acceleration structure property `name`. See `rtAccelerationSetProperty` for a list of supported properties. If the property name is not found, an empty string is returned. `return_string` will be set to point to the returned string. The memory `return_string` points to will be valid until the next API call that returns a string.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetProperty` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationSetProperty, rtAccelerationSetBuilder, rtAccelerationSetTraverser

1.6.8 `rtAccelerationGetTraverser`

NAME

`rtAccelerationGetTraverser` - Query the current traverser from an acceleration structure.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationGetTraverser(RTacceleration acceleration,
                                    char** return_string)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`return_string`

Return string buffer.

DESCRIPTION

`rtAccelerationGetTraverser` returns the name of the traverser currently used in the acceleration structure `acceleration`. If no traverser has been set for `acceleration`, an empty string is returned. `return_string` will be set to point to the returned string. The memory `return_string` points to will be valid until the next API call that returns a string.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationGetTraverser` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationSetTraverser

1.6.9 `rtAccelerationIsDirty`

NAME

`rtAccelerationIsDirty` - Returns the dirty flag of an acceleration structure.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationIsDirty(RTacceleration acceleration,  
                               int* dirty)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`dirty`

Returned dirty flag.

DESCRIPTION

`rtAccelerationIsDirty` returns whether the acceleration structure is currently marked dirty. If the flag is set, a nonzero value will be returned in the location pointed to by **dirty**. Otherwise, zero is returned.

Any acceleration structure which is marked dirty will be rebuilt on a call to one of the **rtContextLaunch** functions, and its dirty flag will be reset. The dirty flag will also be reset on a successful call to **rtAccelerationSetData**.

An acceleration structure which is not marked dirty will never be rebuilt, even if associated groups, geometry, properties, or any other values have changed.

Initially after creation, acceleration structures are marked dirty.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationIsDirty` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationMarkDirty, rtAccelerationSetData, rtContextLaunch

1.6.10 `rtAccelerationMarkDirty`

NAME

`rtAccelerationMarkDirty` - Marks an acceleration structure as dirty.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationMarkDirty(RTacceleration acceleration)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

DESCRIPTION

`rtAccelerationMarkDirty` sets the dirty flag for `acceleration`.

Any acceleration structure which is marked dirty will be rebuilt on a call to one of the `rtContextLaunch` functions, and its dirty flag will be reset. The dirty flag will also be reset on a successful call to `rtAccelerationSetData`.

An acceleration structure which is not marked dirty will never be rebuilt, even if associated groups, geometry, properties, or any other values have changed.

Initially after creation, acceleration structures are marked dirty.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationMarkDirty` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationIsDirty, *rtAccelerationSetData*, *rtContextLaunch*

1.6.11 `rtAccelerationSetBuilder`

NAME

`rtAccelerationSetBuilder` - Specifies the builder to be used for an acceleration structure.

SYNOPSIS

```
#include <optix.h>

Rtresult rtAccelerationSetBuilder(Rtacceleration acceleration,
                                  const char* builder)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`builder`

String value specifying the builder type.

DESCRIPTION

`rtAccelerationSetBuilder` specifies the method used to construct the ray tracing acceleration structure represented by `acceleration`. A builder has to be set for the acceleration structure to pass validation. The current builder can be changed at any time, including after a call to `rtContextLaunch`. In this case, data previously computed for the acceleration structure is invalidated and the acceleration will be marked dirty.

An acceleration structure is only valid with a correct pair of builder and traverser. The traverser type is specified using `rtAccelerationSetTraverser`. For a list of valid combinations of builders and traversers, see below. For a description of the individual traversers, see `rtAccelerationSetTraverser`.

`builder` can take one of the following values:

`"NoAccel"`: Specifies that no acceleration structure is explicitly built. Traversal linearly loops through the list of primitives to intersect. This can be useful e.g. for higher level groups with only few children, where managing a more complex structure introduces unnecessary overhead. Valid traverser types: `"NoAccel"`.

`"Bvh"`: A standard bounding volume hierarchy, useful for most types of graph levels and geometry. Medium build speed, good ray tracing performance. Valid traverser types: `"Bvh"`.

`"Sbvh"`: A high quality BVH variant for maximum ray tracing performance. Slower build speed and slightly higher memory footprint than `"Bvh"`. Valid traverser types: `"Bvh"`.

`"MedianBvh"`: A medium quality bounding volume hierarchy with quick build performance. Useful for dynamic and semi-dynamic content. Valid traverser types: `"Bvh"`.

`"Lbvh"`: A simple bounding volume hierarchy with very fast build performance. Useful for dynamic content. Valid traverser types: `"Bvh"`.

"TriangleKdTree": A high quality kd-tree builder, for triangle geometry only. This may provide better ray tracing performance than the BVH builders for some scenarios. Valid traverser types: *"KdTree"*.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtAccelerationSetBuilder was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationGetBuilder, *rtAccelerationSetTraverser*, *rtAccelerationSetProperty*

1.6.12 `rtAccelerationSetData`

NAME

`rtAccelerationSetData` - Sets the state of an acceleration structure.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationSetData(RTacceleration acceleration,
                               const void* data,
                               RTsize size)
```

PARAMETERS

acceleration

The acceleration structure handle.

data

Pointer to data containing the serialized state.

size

The size in bytes of the buffer pointed to by **data**.

DESCRIPTION

`rtAccelerationSetData` sets the full state of the **acceleration** object, including builder and traverser type as well as properties, as defined by **data**. The memory pointed to by **data** must be unaltered values previously retrieved from a (potentially different) acceleration structure handle. This mechanism is useful for implementing caching mechanisms, especially when using high quality structures which are expensive to build.

Note that no check is performed on whether the contents of **data** match the actual underlying geometry on which the acceleration structure is used. If the children of associated groups or geometry groups differ in number of children, layout of bounding boxes, or geometry, then behavior after this call is undefined.

This call returns `RT_ERROR_VERSION_MISMATCH` if the specified data was retrieved from a different, incompatible version of *OptiX*. In this case, the state of **acceleration** is not changed.

If the call is successful, the dirty flag of **acceleration** will be cleared.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

RT_ERROR_INVALID_VALUE

RT_ERROR_VERSION_MISMATCH

HISTORY

rtAccelerationSetData was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationGetData, *rtAccelerationGetDataSize*

1.6.13 `rtAccelerationSetProperty`

NAME

`rtAccelerationSetProperty` - Sets an acceleration structure property.

SYNOPSIS

```
#include <optix.h>

RTresult rtAccelerationSetProperty(RTacceleration acceleration,
                                   const char* name,
                                   const char* value)
```

PARAMETERS

acceleration

The acceleration structure handle.

name

String value specifying the name of the property.

value

String value specifying the value of the property.

DESCRIPTION

`rtAccelerationSetProperty` sets a named property value for an acceleration structure. Properties can be used to fine tune the way an acceleration structure is built, in order to achieve faster build times or better ray tracing performance. Properties are evaluated and applied by the acceleration structure during build time, and different builders recognize different properties. Setting a property will never fail as long as **acceleration** is a valid handle. Properties that are not recognized by an acceleration structure will be ignored.

The following is a list of the properties used by the individual builders:

"NoAccel": No properties are available for this builder.

"Bvh": *refit* is a flag specifying whether the BVH should be refitted or rebuilt from scratch when a valid BVH over similar geometry is already existent. Refitting is much faster than a full rebuild, and usually yields good ray tracing performance if deformations to the underlying geometry are not too large. The default is 0.

"Sbvh": The SBVH can be used for any type of geometry, but especially efficient structures can be built for triangles. For this case, the following properties are used in order to provide the necessary geometry information to the acceleration object: *vertex_buffer_name* specifies the name of the vertex buffer variable for underlying geometry, containing float3 vertices. *vertex_buffer_stride* is used to define the offset between two vertices in the buffer, given in bytes. The default stride is zero, which assumes that the vertices are tightly packed. *index_buffer_name* specifies the name of the index buffer variable for underlying geometry (if any).

The entries in this buffer are indices of type `int`, where each index refers to one entry in the vertex buffer. A sequence of three indices represent one triangle. *index_buffer_stride* can be used analog to *vertex_buffer_stride* to describe interleaved arrays.

"*MedianBvh*": No properties are available for this builder.

"*Lbvh*": No properties are available for this builder.

"*TriangleKdTree*": Since the kd-tree can build its acceleration structure over triangles only, the geometry data and its format must be made available to the acceleration object. See *Sbvh* for a description of the relevant properties (*vertex_buffer_name*, *index_buffer_name*, *vertex_buffer_stride*, and *index_buffer_stride*).

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationSetProperty` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationGetProperty, *rtAccelerationSetBuilder*, *rtAccelerationSetTraverser*

1.6.14 `rtAccelerationSetTraverser`

NAME

`rtAccelerationSetTraverser` - Specifies the traverser to be used for an acceleration structure.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationSetTraverser(RTacceleration acceleration,
                                   const char* traverser)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

`traverser`

String value specifying the traverser type.

DESCRIPTION

`rtAccelerationSetTraverser` specifies the method used to traverse the ray tracing acceleration structure represented by `acceleration`. A traverser has to be set for the acceleration structure to pass validation. The current active traverser can be changed at any time.

An acceleration structure is only valid with a correct pair of builder and traverser. The builder type is specified using `rtAccelerationSetBuilder`. For a list of valid combinations of builders and traversers, see below. For a description of the individual builders, see `rtAccelerationSetBuilder`.

`traverser` can take one of the following values:

`"NoAccel"`: Linearly loops through the list of primitives to intersect. This is highly inefficient in all but the most trivial scenarios (but there it can provide good performance due to very little overhead). Valid builder types: `"NoAccel"`.

`"Bvh"`: Optimized traversal of generic bounding volume hierarchies. Valid builder types: `"Sbvh"`, `"Bvh"`, `"MedianBvh"`, `"Lbvh"`.

`"BvhCompact"`: Optimized traversal of bounding volume hierarchies for large datasets when virtual memory is turned on. It compresses the BVH data in 4 times before uploading to the device. And decompress the BVH data in real-time during traversal of a bounding volume hierarchy. Valid builder types: `"Bvh"`, `"MedianBvh"`, `"Lbvh"`.

`"KdTree"`: Standard traversal for kd-trees. Valid builder types: `"TriangleKdTree"`.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

`rtAccelerationSetTraverser` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationGetTraverser, *rtAccelerationSetBuilder*, *rtAccelerationSetProperty*

1.6.15 `rtAccelerationValidate`

NAME

`rtAccelerationValidate` - Validates the state of an acceleration structure.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtAccelerationValidate(RTacceleration acceleration)
```

PARAMETERS

`acceleration`

The acceleration structure handle.

DESCRIPTION

`rtAccelerationValidate` checks `acceleration` for completeness. If `acceleration` is not valid, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtAccelerationValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtAccelerationCreate

1.7 Geometry Instance

NAME

Geometry Instance

DESCRIPTION

This section describes the API functions for creation and handling of Geometry instances.

`rtGeometryInstanceCreate`

`rtGeometryInstanceDeclareVariable`

`rtGeometryInstanceDestroy`

`rtGeometryInstanceGetContext`

`rtGeometryInstanceGetGeometry`

`rtGeometryInstanceGetMaterialCount`

`rtGeometryInstanceGetMaterial`

`rtGeometryInstanceGetVariableCount`

`rtGeometryInstanceGetVariable`

`rtGeometryInstanceQueryVariable`

`rtGeometryInstanceRemoveVariable`

`rtGeometryInstanceSetGeometry`

`rtGeometryInstanceSetMaterialCount`

`rtGeometryInstanceSetMaterial`

`rtGeometryInstanceValidate`

HISTORY

Geometry instances were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.7.1 `rtGeometryInstanceCreate`

NAME

`rtGeometryInstanceCreate` - creates a new geometry instance node

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryInstanceCreate(RTcontext context,
                                  RTgeometryinstance* geometryinstance)
```

PARAMETERS

`context`

Specifies the rendering context of the GeometryInstance node.

`geometryinstance`

New GeometryInstance node handle.

DESCRIPTION

`rtGeometryInstanceCreate` creates a new geometry instance node within a context. `context` specifies the target context, and should be a value returned by `rtContextCreate`. After the call, `*geometryinstance` shall be set to the handle of a newly created geometry instance node within `context`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceCreate` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceDestroy, *rtGeometryInstanceDestroy*, *rtGeometryInstanceGetContext*

1.7.2 `rtGeometryInstanceDeclareVariable`

NAME

`rtGeometryInstanceDeclareVariable` - declares a new named variable associated with a geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceDeclareVariable(RTgeometryinstance geometryinstance,
                                           const char* name,
                                           RTvariable* v)
```

PARAMETERS

`geometryinstance`

Specifies the associated `GeometryInstance` node.

`name`

The name that identifies the variable.

`v`

Returns a handle to a newly declared variable.

DESCRIPTION

`rtGeometryInstanceDeclareVariable` declares a new variable associated with a geometry instance node. **`geometryinstance`** specifies the target geometry node, and should be a value returned by **`rtGeometryInstanceCreate`**. **`name`** specifies the name of the variable, and should be a NULL-terminated string. If there is currently no variable associated with **`geometryinstance`** named **`name`**, a new variable named **`name`** will be created and associated with **`geometryinstance`**. After the call, **`*v`** will be set to the handle of the newly-created variable. Otherwise, **`*v`** will be set to NULL. After declaration, the variable can be queried with **`rtGeometryInstanceQueryVariable`** or **`rtGeometryInstanceGetVariable`**. A declared variable does not have a type until its value is set with one of the **`rtVariableSet`** functions. Once a variable is set, its type cannot be changed anymore.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

RT_ERROR_VARIABLE_REDECLARED

RT_ERROR_ILLEGAL_SYMBOL

HISTORY

`rtGeometryInstanceDeclareVariable` was introduced in *OptiX* 1.0.

SEE ALSO

Variables, *rtGeometryInstanceQueryVariable*, *rtGeometryInstanceGetVariable*, *rtGeometryInstanceRemoveVariable*

1.7.3 `rtGeometryInstanceDestroy`

NAME

`rtGeometryInstanceDestroy` - Destroys a geometry instance node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceDestroy(RTgeometryinstance geometryinstance)
```

PARAMETERS

`geometryinstance`

Handle of the geometry instance node to destroy

DESCRIPTION

`rtGeometryInstanceDestroy` removes `geometryinstance` from its context and deletes it. `geometryinstance` should be a value returned by `rtGeometryInstanceCreate`. Associated variables declared via `rtGeometryInstanceDeclareVariable` are destroyed, but no child graph nodes are destroyed. After the call, `geometryinstance` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceCreate

1.7.4 `rtGeometryInstanceGetContext`

NAME

`rtGeometryInstanceGetContext` - returns the context associated with a geometry instance node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceGetContext(RTgeometryinstance geometryinstance,  
                                     RTcontext* context)
```

PARAMETERS

geometryinstance

Specifies the geometry instance.

context

Handle for queried context.

DESCRIPTION

`rtGeometryInstanceGetContext` queries a geometry instance node for its associated context. **geometryinstance** specifies the geometry node to query, and should be a value returned by `rtMGeometryInstanceCreate`. After the call, ***context** shall be set to the context associated with **geometryinstance**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceGetContext

1.7.5 `rtGeometryInstanceGetGeometry`

NAME

`rtGeometryInstanceGetGeometry` - returns the attached Geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceGetGeometry(RTgeometryinstance geometryinstance,  
                                       RTgeometry* geometry)
```

PARAMETERS

`geometryinstance`

GeometryInstance node handle to query geometry.

`geometry`

Handle to attached Geometry node.

DESCRIPTION

`rtGeometryInstanceGetGeometry` sets `geometry` to the handle of the attached Geometry node. If no Geometry node is attached, `RT_ERROR_INVALID_VALUE` is returned, else `RT_SUCCESS`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceGetGeometry` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceCreate, *rtGeometryInstanceDestory*, *rtGeometryInstanceValidate*, *rtGeometryInstanceSetGeometry*

1.7.6 `rtGeometryInstanceGetMaterialCount`

NAME

`rtGeometryInstanceGetMaterialCount` - returns the number of attached materials

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryInstanceGetMaterialCount(RTgeometryinstance geometryinstance,
                                             unsigned int* count)
```

PARAMETERS

`geometryinstance`

GeometryInstance node to query from the number of materials.

`count`

Number of attached materials.

DESCRIPTION

`rtGeometryInstanceGetMaterialCount` returns for `geometryinstance` the number of attached Material nodes `count`. The number of materials can be set with `rtGeometryInstanceSetMaterialCount`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGeometryInstanceGetMaterialCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceSetMaterialCount

1.7.7 `rtGeometryInstanceGetMaterial`

NAME

`rtGeometryInstanceGetMaterial` - returns a material handle

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryInstanceGetMaterial(RTgeometryinstance geometryinstance,
                                       unsigned int idx,
                                       RTmaterial* material)
```

PARAMETERS

`geometryinstance`

GeometryInstance node handle to query material.

`idx`

Index of material.

`material`

Handle to material.

DESCRIPTION

`rtGeometryInstanceGetMaterial` returns handle `material` for the Material node at position `idx` in the material list of `geometryinstance`. `idx` must be in the range of `0` to `rtGeometryInstanceGetMaterialCount()-1`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceGetMaterial` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceGetMaterialCount, rtGeometryInstanceSetMaterial

1.7.8 `rtGeometryInstanceGetVariableCount`

NAME

`rtGeometryInstanceGetVariableCount` - returns the number of attached variables

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceGetVariableCount(RTgeometryinstance geometryinstance,  
                                             unsigned int* count)
```

PARAMETERS

`geometryinstance`

The `GeometryInstance` node to query from the number of attached variables.

`count`

Returns the number of attached variables.

DESCRIPTION

`rtGeometryInstanceGetVariableCount` queries the number of variables attached to a geometry instance. `geometryinstance` specifies the geometry instance, and should be a value returned by `rtGeometryInstanceCreate`. After the call, the number of variables attached to `geometryinstance` is returned to `*count`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceGetVariableCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceSetVariableCount, rtGeometryInstanceDeclareVariable, rtGeometryInstanceRemoveVariable

1.7.9 `rtGeometryInstanceGetVariable`

NAME

`rtGeometryInstanceGetVariable` - returns a handle to an indexed variable of a geometry instance node

SYNOPSIS

```
#include <optix.h>
```

```
RtResult rtGeometryInstanceGetVariable(Rtgeometryinstance geometryinstance,
                                       unsigned int index,
                                       Rtvariable* v)
```

PARAMETERS

`geometryinstance`

The `GeometryInstance` node from which to query a variable.

`index`

The index that identifies the variable to be queried.

`v`

Returns handle to indexed variable.

DESCRIPTION

`rtGeometryInstanceGetVariable` queries the handle of a geometry instance's indexed variable. `geometryinstance` specifies the target geometry instance and should be a value returned by `rtGeometryInstanceCreate`. `index` specifies the index of the variable, and should be a value less than `rtGeometryInstanceGetVariableCount`. If `index` is the index of a variable attached to `geometryinstance`, `*v` will be a handle to that variable after the call. Otherwise, `*v` will be `NULL` after the call. `*v` has to be declared first with `rtGeometryInstanceDeclareVariable` before it can be queried.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtGeometryInstanceGetVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryDeclareVariable, *rtGeometryGetVariableCount*, *rtGeometryRemoveVariable*, *rtGeometryQueryVariable*

1.7.10 `rtGeometryInstanceQueryVariable`

NAME

`rtGeometryInstanceQueryVariable` - returns a handle to a named variable of a geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceQueryVariable(RTgeometryinstance geometryinstance,
                                         const char* name,
                                         RTvariable* v)
```

PARAMETERS

`geometryinstance`

The `GeometryInstance` node to query from a variable.

`name`

The name that identifies the variable to be queried.

`v`

Returns the named variable.

DESCRIPTION

`rtGeometryInstanceQueryVariable` queries the handle of a geometry instance node's named variable. `geometryinstance` specifies the target geometry node and should be a value returned by `rtGeometryInstanceCreate`. `name` specifies the name of the variable, and should be a NULL-terminated string. If `name` is the name of a variable attached to `geometryinstance`, `*v` will be a handle to that variable after the call. Otherwise, `*v` will be NULL after the call. Geometry instance variables have to be declared with `rtGeometryInstanceDeclareVariable` before they can be queried.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceQueryVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceDeclareVariable, *rtGeometryInstanceRemoveVariable*, *rtGeometryInstanceGetVariableCount*, *rtGeometryInstanceGetVariable*

1.7.11 `rtGeometryInstanceRemoveVariable`

NAME

`rtGeometryInstanceRemoveVariable` - removes a named variable from a geometry instance node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceRemoveVariable(RTgeometryinstance geometryinstance,  
                                          RTvariable v)
```

PARAMETERS

`geometryinstance`

The `GeometryInstance` node from which to remove a variable.

`v`

The variable to be removed.

DESCRIPTION

`rtGeometryInstanceRemoveVariable` removes a named variable from a geometry instance. The target geometry instance is specified by `geometryinstance`, which should be a value returned by `rtGeometryInstanceCreate`. The variable to be removed is specified by `v`, which should be a value returned by `rtGeometryInstanceDeclareVariable`. Once a variable has been removed from this geometry instance, another variable with the same name as the removed variable may be declared.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtGeometryInstanceRemoveVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextRemoveVariable, rtGeometryInstanceDeclareVariable

1.7.12 `rtGeometryInstanceSetGeometry`

NAME

`rtGeometryInstanceSetGeometry` - attaches a Geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceSetGeometry(RTgeometryinstance geometryinstance,  
                                       RTgeometry geometry)
```

PARAMETERS

`geometryinstance`

GeometryInstance node handle to attach geometry.

`geometry`

Geometry handle to attach to `geometryinstance`.

DESCRIPTION

`rtGeometryInstanceSetGeometry` attaches a Geometry node to a GeometryInstance. Only **one** Geometry node can be attached to a GeometryInstance. However, it is at any time possible to attach a different Geometry node.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceSetGeometry` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceGetGeometry

1.7.13 `rtGeometryInstanceSetMaterialCount`

NAME

`rtGeometryInstanceSetMaterialCount` - sets the number of materials

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceSetMaterialCount(RTgeometryinstance geometryinstance,  
                                             unsigned int count)
```

PARAMETERS

`geometryinstance`

GeometryInstance node to set number of materials.

`count`

Number of materials to be set.

DESCRIPTION

`rtGeometryInstanceSetMaterialCount` sets the number of materials `count` that will be attached to `geometryinstance`. The number of attached materials can be changed at any time. Increasing the number of materials will not modify already assigned materials. Decreasing the number of materials will not modify the remaining already assigned materials.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceSetMaterialCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceGetMaterialCount

1.7.14 `rtGeometryInstanceSetMaterial`

NAME

`rtGeometryInstanceSetMaterial` - sets a material

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryInstanceSetMaterial(RTgeometryinstance geometryinstance,
                                       unsigned int idx,
                                       RTmaterial material)
```

PARAMETERS

`geometryinstance`

GeometryInstance node for which to set a material.

`idx`

Index into the material list.

`material`

Material handle to attach to `geometryinstance`.

DESCRIPTION

`rtGeometryInstanceSetMaterial` attaches `material` to `geometryinstance` at position `idx` in its internal Material node list. `idx` has to be in the range `0` to `rtGeometryInstanceGetMaterialCount()-1`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceSetMaterial` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceGetMaterialCount, rtGeometryInstanceSetMaterialCount

1.7.15 `rtGeometryInstanceValidate`

NAME

`rtGeometryInstanceValidate` - checks a `GeometryInstance` node for internal consistency

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryInstanceValidate(RTgeometryinstance geometryinstance)
```

PARAMETERS

`geometryinstance`

GeometryInstance node of a model sub-tree to be validated.

DESCRIPTION

`rtGeometryInstanceValidate` checks `geometryinstance` for completeness. If `geometryinstance` or any of the objects attached to `geometry` are not valid, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryInstanceValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryInstanceCreate

1.8 Geometry

NAME

Geometry

DESCRIPTION

This section describes the API functions for creation and handling of Geometry objects.

`rtGeometryCreate`

`rtGeometryDeclareVariable`

`rtGeometryDestroy`

`rtGeometryGetBoundingBoxProgram`

`rtGeometryGetContext`

`rtGeometryGetIntersectionProgram`

`rtGeometryGetPrimitiveCount`

`rtGeometryGetVariableCount`

`rtGeometryGetVariable`

`rtGeometryIsDirty`

`rtGeometryMarkDirty`

`rtGeometryQueryVariable`

`rtGeometryRemoveVariable`

`rtGeometrySetBoundingBoxProgram`

`rtGeometrySetIntersectionProgram`

`rtGeometrySetPrimitiveCount`

`rtGeometryValidate`

HISTORY

Geometry objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Material, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.8.1 rtGeometryCreate

NAME

rtGeometryCreate - creates a new geometry node

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryCreate(RTcontext context,
                          RTgeometry* geometry)
```

PARAMETERS

context

Specifies the rendering context of the Geometry node.

geometry

New Geometry node handle.

DESCRIPTION

rtGeometryCreate creates a new geometry node within a context. **context** specifies the target context, and should be a value returned by **rtContextCreate**. After the call, ***geometry** shall be set to the handle of a newly created geometry node within **context**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtGeometryCreate was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryDestroy, *rtGeometrySetBoundingBoxProgram*, *rtGeometrySetIntersectionProgram*

1.8.2 rtGeometryDeclareVariable

NAME

rtGeometryDeclareVariable - declares a new named variable associated with a geometry instance

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryDeclareVariable(RTgeometry geometry,
                                   const char* name,
                                   RTvariable* v)
```

PARAMETERS

geometry

Specifies the associated Geometry node.

name

The name that identifies the variable.

v

Returns a handle to a newly declared variable.

DESCRIPTION

rtGeometryDeclareVariable declares a new variable associated with a geometry node. **geometry** specifies the target geometry node, and should be a value returned by **rtGeometryCreate**. **name** specifies the name of the variable, and should be a NULL-terminated string. If there is currently no variable associated with **geometry** named **name**, a new variable named **name** will be created and associated with **geometry**. After the call, ***v** will be set to the handle of the newly-created variable. Otherwise, ***v** will be set to NULL. After declaration, the variable can be queried with **rtGeometryQueryVariable** or **rtGeometryGetVariable**. A declared variable does not have a type until its value is set with one of the **rtVariableSet** functions. Once a variable is set, its type cannot be changed anymore.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_VARIABLE_REDECLARED

RT_ERROR_ILLEGAL_SYMBOL

HISTORY

`rtGeometryDeclareVariable` was introduced in *OptiX* 1.0.

SEE ALSO

Variables, rtGeometryQueryVariable, rtGeometryGetVariable, rtGeometryRemoveVariable

1.8.3 `rtGeometryDestroy`

NAME

`rtGeometryDestroy` - Destroys a geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryDestroy(RTgeometry geometry)
```

PARAMETERS

`geometry`

Handle of the geometry node to destroy

DESCRIPTION

`rtGeometryDestroy` removes `geometry` from its context and deletes it. `geometry` should be a value returned by `rtGeometryCreate`. Associated variables declared via `rtGeometryDeclareVariable` are destroyed, but no child graph nodes are destroyed. After the call, `geometry` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryCreate, *rtGeometrySetPrimitiveCount*, *rtGeometryGetPrimitiveCount*

1.8.4 `rtGeometryGetBoundingBoxProgram`

NAME

`rtGeometryGetBoundingBoxProgram` - returns the attached bounding box program

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGetBoundingBoxProgram(RTgeometry geometry,  
                                         RTprogram* program)
```

PARAMETERS

`geometry`

Geometry node handle from which to query program.

`program`

Handle to attached bounding box program.

DESCRIPTION

`rtGeometryGetBoundingBoxProgram` returns the handle `program` for the attached bounding box program of `geometry`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGetBoundingBoxProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometrySetBoundingBoxProgram

1.8.5 `rtGeometryGetContext`

NAME

`rtGeometryGetContext` - returns the context associated with a geometry node

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGetContext(RTgeometry geometry,
                              RTcontext* context)
```

PARAMETERS

geometry

Specifies the geometry to query.

context

The context associated with **geometry**.

DESCRIPTION

`rtGeometryGetContext` queries a geometry node for its associated context. **geometry** specifies the geometry node to query, and should be a value returned by `rtMGeometryCreate`. After the call, ***context** shall be set to the context associated with **geometry**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryCreate

1.8.6 `rtGeometryGetIntersectionProgram`

NAME

`rtGeometryGetIntersectionProgram` - returns the attached intersection program

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryGetIntersectionProgram(RTgeometry geometry,  
                                         RTprogram* program)
```

PARAMETERS

`geometry`

Geometry node handle to query program.

`program`

Handle to attached intersection program.

DESCRIPTION

`rtGeometryGetIntersectionProgram` returns in `program` a handle of the attached intersection program.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGetIntersectionProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometrySetIntersectionProgram, *rtProgramCreateFromPTXFile*, *rtProgramCreateFromPTXString*

1.8.7 `rtGeometryGetPrimitiveCount`

NAME

`rtGeometryGetPrimitiveCount` - returns the number of primitives

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGetPrimitiveCount(RTgeometry geometry,
                                     unsigned int* num_primitives)
```

PARAMETERS

`geometry`

Geometry node to query from the number of primitives.

`num_primitives`

Number of primitives.

DESCRIPTION

`rtGeometryGetPrimitiveCount` returns for `geometry` the number of set primitives. The number of primitives can be set with `rtGeometrySetPrimitiveCount`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGetPrimitiveCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometrySetPrimitiveCount

1.8.8 `rtGeometryGetVariableCount`

NAME

`rtGeometryGetVariableCount` - returns the number of attached variables

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGetVariableCount(RTgeometry geometry,
                                     unsigned int* count)
```

PARAMETERS

`geometry`

The Geometry node to query from the number of attached variables.

`count`

Returns the number of attached variables.

DESCRIPTION

`rtGeometryGetVariableCount` queries the number of variables attached to a geometry node. `geometry` specifies the geometry node, and should be a value returned by `rtGeometryCreate`. After the call, the number of variables attached to `geometry` is returned to `*count`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryGetVariableCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometrySetVariableCount, *rtGeometryDeclareVariable*, *rtGeometryRemoveVariable*

1.8.9 `rtGeometryGetVariable`

NAME

`rtGeometryGetVariable` - returns a handle to an indexed variable of a geometry node

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryGetVariable(RTgeometry geometry,
                               unsigned int index,
                               RTvariable* v)
```

PARAMETERS

`geometry`

The geometry node from which to query a variable.

`index`

The index that identifies the variable to be queried.

`v`

Returns handle to indexed variable.

DESCRIPTION

`rtGeometryGetVariable` queries the handle of a geometry node's indexed variable. `geometry` specifies the target geometry and should be a value returned by `rtGeometryCreate`. `index` specifies the index of the variable, and should be a value less than `rtGeometryGetVariableCount`. If `index` is the index of a variable attached to `geometry`, `*v` will be a handle to that variable after the call. Otherwise, `*v` will be NULL after the call. `*v` has to be declared first with `rtGeometryDeclareVariable` before it can be queried.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtGeometryGetVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryDeclareVariable, *rtGeometryGetVariableCount*, *rtGeometryRemoveVariable*, *rtGeometryQueryVariable*

1.8.10 `rtGeometryIsDirty`

NAME

`rtGeometryIsDirty` - returns the dirty flag

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryIsDirty(RTgeometry geometry,  
                           int* dirty)
```

PARAMETERS

`geometry`

The geometry node to query from the dirty flag.

`dirty`

Dirty flag.

DESCRIPTION

`rtGeometryIsDirty` returns the dirty flag of `geometry`. The dirty flag for geometry nodes can be set with `rtGeometryMarkDirty`. By default the flag is **1** for a new geometry node, indicating dirty. After a call to `rtContextLaunch` the flag is automatically set to **0**. When the dirty flag is set, the geometry data is uploaded automatically to the device while a `rtContextLaunch` call.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryIsDirty` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextLaunch, rtGeometryMarkDirty

1.8.11 `rtGeometryMarkDirty`

NAME

`rtGeometryMarkDirty` - sets the dirty flag

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryMarkDirty(RTgeometry geometry)
```

PARAMETERS

`geometry`

The geometry node to mark as dirty.

DESCRIPTION

`rtGeometryMarkDirty` sets for `geometry` the dirty flag. By default the dirty flag is set for a new Geometry node. After a call to `rtContextLaunch` the flag is automatically cleared. When the dirty flag is set, the geometry data is uploaded automatically to the device while a `rtContextLaunch` call.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryMarkDirty` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryIsDirty

1.8.12 `rtGeometryQueryVariable`

NAME

`rtGeometryQueryVariable` - returns a handle to a named variable of a geometry node

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometryQueryVariable(RTgeometry geometry,
                                const char* name,
                                RTvariable* v)
```

PARAMETERS

`geometry`

The geometry node to query from a variable.

`name`

The name that identifies the variable to be queried.

`v`

Returns the named variable.

DESCRIPTION

`rtGeometryQueryVariable` queries the handle of a geometry node's named variable. `geometry` specifies the target geometry node and should be a value returned by `rtGeometryCreate`. `name` specifies the name of the variable, and should be a NULL-terminated string. If `name` is the name of a variable attached to `geometry`, `*v` will be a handle to that variable after the call. Otherwise, `*v` will be NULL after the call. Geometry variables have to be declared with `rtGeometryDeclareVariable` before they can be queried.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtGeometryQueryVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryDeclareVariable, *rtGeometryRemoveVariable*, *rtGeometryGetVariableCount*, *rtGeometryGetVariable*

1.8.13 `rtGeometryRemoveVariable`

NAME

`rtGeometryRemoveVariable` - removes a named variable from a geometry node

SYNOPSIS

```
#include <optix.h>
```

```
RtResult rtGeometryRemoveVariable(Rtgeometry geometry,  
                                  Rtvariable v)
```

PARAMETERS

`geometry`

The geometry node from which to remove a variable.

`v`

The variable to be removed.

DESCRIPTION

`rtGeometryRemoveVariable` removes a named variable from a geometry node. The target geometry is specified by `geometry`, which should be a value returned by `rtGeometryCreate`. The variable to remove is specified by `v`, which should be a value returned by `rtGeometryDeclareVariable`. Once a variable has been removed from this geometry node, another variable with the same name as the removed variable may be declared.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtGeometryRemoveVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextRemoveVariable

1.8.14 `rtGeometrySetBoundingBoxProgram`

NAME

`rtGeometrySetBoundingBoxProgram` - sets the bounding box program

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometrySetBoundingBoxProgram(RTgeometry geometry,  
                                         RTprogram program)
```

`geometry`

The geometry node for which to set the bounding box program.

`program`

Handle to the bounding box program.

DESCRIPTION

`rtGeometrySetBoundingBoxProgram` sets for `geometry` the `program` that computes an axis aligned bounding box for each attached primitive to `geometry`. `RTprogram`'s can be either generated with `rtProgramCreateFromPTXFile` or `rtProgramCreateFromPTXString`. A bounding box program is mandatory for every geometry node.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtGeometrySetBoundingBoxProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGetBoundingBoxProgram, *rtProgramCreateFromPTXFile*, *rtProgramCreateFromPTXString*

1.8.15 `rtGeometrySetIntersectionProgram`

NAME

`rtGeometrySetIntersectionProgram` - sets the intersection program

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometrySetIntersectionProgram(RTgeometry geometry,  
                                          RTprogram program)
```

PARAMETERS

`geometry`

The geometry node for which to set the intersection program.

`program`

A handle to the ray primitive intersection program.

DESCRIPTION

`rtGeometrySetIntersectionProgram` sets for `geometry` the `program` that performs ray primitive intersections. `RTprogram`'s can be either generated with `rtProgramCreateFromPTXFile` or `rtProgramCreateFromPTXString`. An intersection program is mandatory for every geometry node.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtGeometrySetIntersectionProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGetIntersectionProgram, rtProgramCreateFromPTXFile, rtProgramCreateFromPTXString

1.8.16 `rtGeometrySetPrimitiveCount`

NAME

`rtGeometrySetPrimitiveCount` - sets the number of primitives

SYNOPSIS

```
#include <optix.h>

RTresult rtGeometrySetPrimitiveCount(RTgeometry geometry,
                                     unsigned int num_primitives)
```

PARAMETERS

`geometry`

The geometry node for which to set the number of primitives.

`num_primitives`

The number of primitives.

DESCRIPTION

`rtGeometrySetPrimitiveCount` sets the number of primitives `num_primitives` in `geometry`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometrySetPrimitiveCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometryGetPrimitiveCount

1.8.17 `rtGeometryValidate`

NAME

`rtGeometryValidate` - validates the geometry nodes integrity

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtGeometryValidate(RTgeometry geometry)
```

PARAMETERS

`geometry`

The geometry node to be validated.

DESCRIPTION

`rtGeometryValidate` checks `geometry` for completeness. If `geometry` or any of the objects attached to `geometry` are not valid, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtGeometryValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextValidate

1.9 Material

NAME

Material

DESCRIPTION

This section describes the API functions for creation and handling of Material objects.

rtMaterialCreate

rtMaterialDeclareVariable

rtMaterialDestroy

rtMaterialGetAnyHitProgram

rtMaterialGetClosestHitProgram

rtMaterialGetContext

rtMaterialGetVariableCount

rtMaterialGetVariable

rtMaterialQueryVariable

rtMaterialRemoveVariable

rtMaterialSetAnyHitProgram

rtMaterialSetClosestHitProgram

rtMaterialValidate

HISTORY

Material objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Program, Buffer, Texture Sampler, Variables, Context-Free Functions

1.9.1 rtMaterialCreate

NAME

rtMaterialCreate - Creates a new material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialCreate(RTcontext context,
                        RTmaterial* material)
```

PARAMETERS

context

Specifies a context within which to create a new material.

material

Returns a newly created material.

DESCRIPTION

rtMaterialCreate creates a new material within a context. **context** specifies the target context, and should be a value returned by **rtContextCreate**. After the call, if **material** is not NULL, ***material** shall be set to the handle of a newly created material within **context**. Otherwise, this call has no effect and returns **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtMaterialCreate was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialDestroy, *rtContextCreate*

1.9.2 rtMaterialDeclareVariable

NAME

rtMaterialDeclareVariable - Declares a new named variable to be associated with a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialDeclareVariable(RTmaterial material,
                                   const char* name,
                                   RTvariable* variable)
```

PARAMETERS

material

Specifies the material to modify.

name

Specifies the name of the variable.

variable

Returns a handle to a newly declared variable.

DESCRIPTION

rtMaterialDeclareVariable declares a new variable to be associated with a material. **material** specifies the target material, and should be a value returned by **rtMaterialCreate**. **name** specifies the name of the variable, and should be a NULL-terminated string. If there is currently no variable associated with **material** named **name**, and **variable** is not NULL, a new variable named **name** will be created and associated with **material**. After the call, ***variable** shall be set to the handle of the newly-created variable. Otherwise, this call has no effect and shall return either **RT_ERROR_INVALID_VALUE** if either **name** or **variable** is equal to NULL or **RT_ERROR_VARIABLE_REDECLARED** if **name** is the name of an existing variable associated with the material.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_VARIABLE_REDECLARED

RT_ERROR_ILLEGAL_SYMBOL

HISTORY

`rtMaterialDeclareVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialGetVariable, *rtMaterialQueryVariable*, *rtMaterialCreate*

1.9.3 rtMaterialDestroy

NAME

rtMaterialDestroy - Destroys a material object

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialDestroy(RTmaterial material)
```

PARAMETERS

material

Handle of the material node to destroy

DESCRIPTION

rtMaterialDestroy removes **material** from its context and deletes it. **material** should be a value returned by **rtMaterialCreate**. Associated variables declared via **rtMaterialDeclareVariable** are destroyed, but no child graph nodes are destroyed. After the call, **material** is no longer a valid handle.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtMaterialDestroy was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialCreate

1.9.4 `rtMaterialGetAnyHitProgram`

NAME

`rtMaterialGetAnyHitProgram` - Returns the any hit program associated with a (material, ray type) tuple.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialGetAnyHitProgram(RTmaterial material,
                                   unsigned int ray_type_index,
                                   RTprogram* program)
```

PARAMETERS

`material`

Specifies the material of the (material, ray type) tuple to query.

`ray_type_index`

Specifies the type of ray of the (material, ray type) tuple to query.

`program`

Returns the any hit program associated with the (material, ray type) tuple.

DESCRIPTION

`rtMaterialGetAnyHitProgram` queries the any hit program associated with a (material, ray type) tuple. `material` specifies the material of interest and should be a value returned by `rtMaterialCreate`. `ray_type_index` specifies the target ray type and should be a value less than the value returned by `rtContextGetRayTypeCount`. After the call, if all parameters are valid, `*program` shall be set to the handle of the any hit program associated with the tuple (`material`, `ray_type_index`). Otherwise, the call has no effect and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtMaterialGetAnyHitProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialSetAnyHitProgram, rtMaterialCreate, rtContextGetRayTypeCount

1.9.5 rtMaterialGetClosestHitProgram

NAME

rtMaterialGetClosestHitProgram - Returns the closest hit program associated with a (material, ray type) tuple.

SYNOPSIS

```
#include <optix.h>
```

```
Rtresult rtMaterialGetClosestHitProgram(Rtmaterial material,
                                       unsigned int ray_type_index,
                                       Rtprogram* program)
```

PARAMETERS

material

Specifies the material of the (material, ray type) tuple to query.

ray_type_index

Specifies the type of ray of the (material, ray type) tuple to query.

program

Returns the closest hit program associated with the (material, ray type) tuple.

DESCRIPTION

rtMaterialGetClosestHitProgram queries the closest hit program associated with a (material, ray type) tuple. **material** specifies the material of interest and should be a value returned by **rtMaterialCreate**. **ray_type_index** specifies the target ray type and should be a value less than the value returned by **rtContextGetRayTypeCount**. After the call, if all parameters are valid, ***program** shall be set to the handle of the any hit program associated with the tuple (**material**, **ray_type_index**). Otherwise, the call has no effect and returns **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtMaterialGetClosestHitProgram was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialSetClosestHitProgram, rtMaterialCreate, rtContextGetRayTypeCount

1.9.6 rtMaterialGetContext

NAME

rtMaterialGetContext - Returns the context associated with a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialGetContext(RTmaterial material,
                             RTcontext* context)
```

PARAMETERS

material

Specifies the material to query.

context

Returns the context associated with the material.

DESCRIPTION

rtMaterialGetContext queries a material for its associated context. **material** specifies the material to query, and should be a value returned by **rtMaterialCreate**. After the call, if both parameters are valid, ***context** shall be set to the context associated with **material**. Otherwise, the call has no effect and returns **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtMaterialGetContext was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialCreate

1.9.7 `rtMaterialGetVariableCount`

NAME

`rtMaterialGetVariableCount` - Returns the number of variables attached to a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialGetVariableCount(RTmaterial material,
                                    unsigned int* count)
```

PARAMETERS

`material`

Specifies the material to query.

`count`

Returns the number of variables.

DESCRIPTION

`rtMaterialGetVariableCount` queries the number of variables attached to a material. `material` specifies the material, and should be a value returned by `rtMaterialCreate`. After the call, if both parameters are valid, the number of variables attached to `material` is returned to `*count`. Otherwise, the call has no effect and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtMaterialGetVariableCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialCreate

1.9.8 `rtMaterialGetVariable`

NAME

`rtMaterialGetVariable` - Returns a handle to an indexed variable of a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialGetVariable(RTmaterial material,
                               unsigned int index,
                               RTvariable *variable)
```

PARAMETERS

material

Specifies the material to query.

index

Specifies the index of the variable to query.

variable

Returns the indexed variable.

DESCRIPTION

`rtMaterialGetVariable` queries the handle of a material's indexed variable. **material** specifies the target material and should be a value returned by `rtMaterialCreate`. **index** specifies the index of the variable, and should be a value less than `rtMaterialGetVariableCount`. If **material** is a valid material and **index** is the index of a variable attached to **material**, ***variable** shall be set to a handle to that variable after the call. Otherwise, ***variable** shall be set to NULL and either **RT_ERROR_INVALID_VALUE** or **RT_ERROR_VARIABLE_NOT_FOUND** shall be returned depending on the validity of **material**, or **index**, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtMaterialGetVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialQueryVariable, *rtMaterialGetVariableCount*, *rtMaterialCreate*

1.9.9 rtMaterialQueryVariable

NAME

rtMaterialQueryVariable - Queries for the existence of a named variable of a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialQueryVariable(RTmaterial material,
                                const char* name,
                                RTvariable* variable)
```

PARAMETERS

material

Specifies the material to query.

name

Specifies the name of the variable to query.

variable

Returns a the named variable, if it exists.

DESCRIPTION

rtMaterialQueryVariable queries for the existence of a material's named variable. **material** specifies the target material and should be a value returned by **rtMaterialCreate**. **name** specifies the name of the variable, and should be a NULL-terminated string. If **material** is a valid material and **name** is the name of a variable attached to **material**, ***variable** shall be set to a handle to that variable after the call. Otherwise, ***variable** shall be set to NULL. If **material** is not a valid material, **RT_ERROR_INVALID_VALUE** shall be returned.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtMaterialQueryVariable was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialGetVariable, rtMaterialCreate

1.9.10 `rtMaterialRemoveVariable`

NAME

`rtMaterialRemoveVariable` - Removes a variable from a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialRemoveVariable(RTmaterial material,
                                  RTvariable variable)
```

PARAMETERS

material

Specifies the material to modify.

variable

Specifies the variable to remove.

DESCRIPTION

`rtMaterialRemoveVariable` removes a variable from a material. The material of interest is specified by **material**, which should be a value returned by `rtMaterialCreate`. The variable to remove is specified by **variable**, which should be a value returned by `rtMaterialDeclareVariable`. Once a variable has been removed from this material, another variable with the same name as the removed variable may be declared. If **material** does not refer to a valid material, this call has no effect and returns `RT_ERROR_INVALID_VALUE`. If **variable** is not a valid variable or does not belong to **material**, this call has no effect and returns `RT_ERROR_INVALID_VALUE` or `RT_ERROR_VARIABLE_NOT_FOUND`, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtMaterialRemoveVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialDeclareVariable, rtMaterialCreate

1.9.11 `rtMaterialSetAnyHitProgram`

NAME

`rtMaterialSetAnyHitProgram` - Sets the any hit program associated with a (material, ray type) tuple.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialSetAnyHitProgram(RTmaterial material,
                                   unsigned int ray_type_index,
                                   RTprogram program)
```

PARAMETERS

`material`

Specifies the material of the (material, ray type) tuple to modify.

`ray_type_index`

Specifies the type of ray of the (material, ray type) tuple to modify.

`program`

Specifies the any hit program to associate with the (material, ray type) tuple.

DESCRIPTION

`rtMaterialSetAnyHitProgram` specifies an any hit program to associate with a (material, ray type) tuple. `material` specifies the target material and should be a value returned by `rtMaterialCreate`. `ray_type_index` specifies the type of ray to which the program applies and should be a value less than the value returned by `rtContextGetRayTypeCount`. `program` specifies the target any hit program which shall apply to the tuple (`material`, `ray_type_index`) and should be a value returned by either `rtProgramCreateFromPTXString` or `rtProgramCreateFromPTXFile`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtMaterialSetAnyHitProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialGetAnyHitProgram, *rtMaterialCreate*, *rtContextGetRayTypeCount*, *rtProgramCreateFromPTXString*, *rtProgramCreateFromPTXFile*

1.9.12 `rtMaterialSetClosestHitProgram`

NAME

`rtMaterialSetClosestHitProgram` - Sets the closest hit program associated with a (material, ray type) tuple.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialSetClosestHitProgram(RTmaterial material,
                                       unsigned int ray_type_index,
                                       RTprogram program)
```

PARAMETERS

`material`

Specifies the material of the (material, ray type) tuple to modify.

`ray_type_index`

Specifies the ray type of the (material, ray type) tuple to modify.

`program`

Specifies the closest hit program to associate with the (material, ray type) tuple.

DESCRIPTION

`rtMaterialSetClosestHitProgram` specifies a closest hit program to associate with a (material, ray type) tuple. `material` specifies the material of interest and should be a value returned by `rtMaterialCreate`. `ray_type_index` specifies the type of ray to which the program applies and should be a value less than the value returned by `rtContextGetRayTypeCount`. `program` specifies the target closest hit program which shall apply to the tuple (`material`, `ray_type_index`) and should be a value returned by either `rtProgramCreateFromPTXString` or `rtProgramCreateFromPTXFile`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtMaterialSetClosestHitProgram` was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialGetClosestHitProgram, *rtMaterialCreate*, *rtContextGetRayTypeCount*, *rtProgramCreateFromPTXString*, *rtProgramCreateFromPTXFile*

1.9.13 rtMaterialValidate

NAME

rtMaterialValidate - Verifies the state of a material.

SYNOPSIS

```
#include <optix.h>

RTresult rtMaterialValidate(RTmaterial material)
```

PARAMETERS

material

Specifies the material to be validated.

DESCRIPTION

rtMaterialValidate checks **material** for completeness. If **material** or any of the objects attached to **material** are not valid, the call will return **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

rtMaterialValidate was introduced in *OptiX* 1.0.

SEE ALSO

rtMaterialCreate

1.10 Program

NAME

Program

DESCRIPTION

This section describes the API functions for creation and handling of program objects.

rtProgramCreateFromPTXFile

rtProgramCreateFromPTXString

rtProgramDeclareVariable

rtProgramDestroy

rtProgramGetContext

rtProgramGetVariableCount

rtProgramGetVariable

rtProgramQueryVariable

rtProgramRemoveVariable

rtProgramValidate

HISTORY

Program objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Buffer, Texture Sampler, Variables, Context-Free Functions

1.10.1 `rtProgramCreateFromPTXFile`

NAME

`rtProgramCreateFromPTXFile` - Creates a new program object.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtProgramCreateFromPTXFile(RTcontext context,  
                                   const char* filename,  
                                   const char* program_name,  
                                   RTprogram* program)
```

PARAMETERS

`context`

The context to create the program in.

`filename`

Path to the file containing the PTX code.

`program_name`

The name of the PTX function to create the program from.

`program`

Handle to the program to be created.

DESCRIPTION

`rtProgramCreateFromPTXFile` allocates and returns a handle to a new program object. The program is created from PTX code held in `filename` from function `program_name`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_INVALID_SOURCE`

`RT_ERROR_FILE_NOT_FOUND`

HISTORY

`rtProgramCreateFromPTXFile` was introduced in *OptiX* 1.0.

SEE ALSO

`RT_PROGRAM`, *rtProgramCreateFromPTXString*, *rtProgramDestroy*

1.10.2 `rtProgramCreateFromPTXString`

NAME

`rtProgramCreateFromPTXString` - Creates a new program object.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtProgramCreateFromPTXString(RTcontext context,  
                                     const char* ptx,  
                                     const char* program_name,  
                                     RTprogram* program)
```

PARAMETERS

`context`

The context to create the program in.

`ptx`

The string containing the PTX code.

`program_name`

The name of the PTX function to create the program from.

`program`

Handle to the program to be created.

DESCRIPTION

`rtProgramCreateFromPTXString` allocates and returns a handle to a new program object. The program is created from PTX code held in the NULL-terminated string `ptx` from function `program_name`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_INVALID_SOURCE`

HISTORY

`rtProgramCreateFromPTXString` was introduced in *OptiX* 1.0.

SEE ALSO

`RT_PROGRAM`, *rtProgramCreateFromPTXFile*, *rtProgramDestroy*

1.10.3 `rtProgramDeclareVariable`

NAME

`rtProgramDeclareVariable` - Declares a new named variable associated with a program.

SYNOPSIS

```
#include <optix.h>

RTresult rtProgramDeclareVariable(RTprogram program,
                                  const char* name,
                                  RTvariable* v)
```

PARAMETERS

`program`

The program the declared variable will be attached to.

`name`

The name of the variable to be created.

`v`

Return handle to the variable to be created.

DESCRIPTION

`rtProgramDeclareVariable` declares a new variable, **name**, and associates it with the program. A variable can only be declared with the same name once on the program; any attempt to declare multiple variables with the same name will cause the call to fail and return **RT_ERROR_VARIABLE_REDECLARED**. If **v** is **NULL** the call will return **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

RT_ERROR_VARIABLE_REDECLARED

RT_ERROR_ILLEGAL_SYMBOL

HISTORY

`rtProgramDeclareVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramRemoveVariable, *rtProgramGetVariable*, *rtProgramGetVariableCount*, *rtProgramQueryVariable*

1.10.4 `rtProgramDestroy`

NAME

`rtProgramDestroy` - Destroys a program object

SYNOPSIS

```
#include <optix.h>

RTresult rtProgramDestroy(RTprogram program)
```

PARAMETERS

`program`

Handle of the program to destroy

DESCRIPTION

`rtProgramDestroy` removes `program` from its context and deletes it. `program` should be a value returned by `rtProgramCreate`. Associated variables declared via `rtProgramDeclareVariable` are destroyed. After the call, `program` is no longer a valid handle.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtProgramDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramCreateFromPTXFile, *rtProgramCreateFromPTXString*

1.10.5 rtProgramGetContext

NAME

rtProgramGetContext - Gets the context object that created a program.

SYNOPSIS

```
#include <optix.h>

RTresult rtProgramGetContext(RTprogram program,
                             RTcontext* context)
```

PARAMETERS

program

The program to be queried for its context object.

context

The return handle for the requested context object.

DESCRIPTION

rtProgramGetContext returns a handle to the context object that was used to create **program**. If **context** is NULL, the call will return **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtProgramGetContext was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate

1.10.6 `rtProgramGetVariableCount`

NAME

`rtProgramGetVariableCount` - Returns the number of variables attached to a program.

SYNOPSIS

```
#include <optix.h>

RTresult rtProgramGetVariableCount(RTprogram program,
                                   unsigned int* count)
```

PARAMETERS

`program`

The program to be queried for its variable count.

`context`

The return handle for the number of variables attached to this program.

DESCRIPTION

`rtProgramGetVariableCount` returns, in `*count`, the number of variable objects that have been attached to `program`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtProgramGetVariableCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramDeclareVariable, *rtProgramRemoveVariable*, *rtProgramGetVariable*, *rtProgramQueryVariable*

1.10.7 `rtProgramGetVariable`

NAME

`rtProgramGetVariable` - Returns a handle to a variable attached to a program by index.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtProgramGetVariable(RTprogram program,  
                             unsigned int index,  
                             RTvariable* v)
```

PARAMETERS

program

The program to be queried for the indexed variable object.

index

The index of the variable to return.

v

Return handle to the variable object specified by the index.

DESCRIPTION

`rtProgramGetVariable` returns a handle to a variable in `*v` attached to `program` with `rtProgramDeclareVariable` by `index`. `index` must be between 0 and one less than the value returned by `rtProgramGetVariableCount`. The order in which variables are enumerated is not constant and may change as variables are attached and removed from the program object.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtProgramGetVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramDeclareVariable, *rtProgramRemoveVariable*, *rtProgramGetVariableCount*, *rtProgramQueryVariable*

1.10.8 rtProgramQueryVariable

NAME

rtProgramQueryVariable - Returns a handle to the named variable attached to a program.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtProgramQueryVariable(RTprogram program,  
                                const char* name,  
                                RTvariable* v)
```

PARAMETERS

program

The program to be queried for the named variable.

name

The name of the program to be queried for.

v

The return handle to the variable object.

program

Handle to the program to be created.

DESCRIPTION

rtProgramQueryVariable returns a handle to a variable object, in ***v**, attached to **program** referenced by the NULL-terminated string **name**. If **name** is not the name of a variable attached to **program**, ***v** will be NULL after the call.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtProgramQueryVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramDeclareVariable, *rtProgramRemoveVariable*, *rtProgramGetVariable*, *rtProgramGetVariableCount*

1.10.9 `rtProgramRemoveVariable`

NAME

`rtProgramRemoveVariable` - Removes the named variable from a program.

SYNOPSIS

```
#include <optix.h>

RTresult rtProgramRemoveVariable(RTprogram program,
                                RTvariable v)
```

PARAMETERS

`program`

The program to remove the variable from.

`v`

The variable to remove.

DESCRIPTION

`rtProgramRemoveVariable` removes variable `v` from the `program` object. Once a variable has been removed from this program, another variable with the same name as the removed variable may be declared.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_VARIABLE_NOT_FOUND`

HISTORY

`rtProgramRemoveVariable` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramDeclareVariable, *rtProgramGetVariable*, *rtProgramGetVariableCount*, *rtProgramQueryVariable*

1.10.10 `rtProgramValidate`

NAME

`rtProgramValidate` - Validates the state of a program.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtProgramValidate(RTprogram program)
```

PARAMETERS

`program`

The program to be validated.

DESCRIPTION

`rtProgramValidate` checks `program` for completeness. If `program` or any of the objects attached to `program` are not valid, the call will return `RT_ERROR_INVALID_CONTEXT`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtProgramValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtProgramCreateFromPTXFile, *rtProgramCreateFromPTXString*

1.11 Buffer

NAME

Buffer

DESCRIPTION

This section describes the API functions for creation and handling of Buffer objects.

`rtBufferCreate`

`rtBufferCreateFromD3D9Resource`

`rtBufferCreateFromD3D10Resource`

`rtBufferCreateFromD3D11Resource`

`rtBufferCreateFromGLBO`

`rtBufferD3D9Unregister`

`rtBufferD3D10Unregister`

`rtBufferD3D11Unregister`

`rtBufferD3D9Register`

`rtBufferD3D10Register`

`rtBufferD3D11Register`

`rtBufferDestroy`

`rtBufferGetContext`

`rtBufferGetD3D9Resource`

`rtBufferGetD3D10Resource`

`rtBufferGetD3D11Resource`

`rtBufferGetDimensionality`

`rtBufferGetElementSize`

`rtBufferGetFormat`

`rtBufferGetGLBOId`

`rtBufferGetSize1D`

`rtBufferGetSize2D`

`rtBufferGetSize3D`

`rtBufferGetSizev`

`rtBufferGLUnregister`

rtBufferGLRegister

rtBufferMap

rtBufferSetElementSize

rtBufferSetSize1D

rtBufferSetSize2D

rtBufferSetSize3D

rtBufferSetSizev

rtBufferUnmap

rtBufferValidate

HISTORY

Buffer objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Texture Sampler, Variables, Context-Free Functions

1.11.1 rtBufferCreate

NAME

rtBufferCreate - Creates a new buffer object.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferCreate(RTcontext context,
                       unsigned int bufferdesc,
                       RTbuffer* buffer)
```

PARAMETERS

context

The context to create the buffer in.

bufferdesc

Bitwise or combination of the **type** and **flags** of the new buffer.

buffer

The return handle for the buffer object.

DESCRIPTION

rtBufferCreate allocates and returns a new handle to a new buffer object in ***buffer** associated with **context**. The backing storage of the buffer is managed by *OptiX*. A buffer is specified by a bitwise or combination of a **type** and **flags** in **bufferdesc**. The supported types are:

```
RT_BUFFER_INPUT
RT_BUFFER_OUTPUT
RT_BUFFER_INPUT_OUTPUT
```

The supported flags are:

```
RT_BUFFER_GPU_LOCAL
```

The type values are used to specify the direction of data flow from the host to the *OptiX* devices. **RT_BUFFER_INPUT** specifies that the host may only write to the buffer and the device may only read from the buffer. **RT_BUFFER_OUTPUT** specifies the opposite, read only access on the host and write only access on the device. Devices and the host may read and write from buffers of type **RT_BUFFER_INPUT_OUTPUT**. Reading or writing to a buffer of the incorrect type (e.g., the host writing to a buffer of type **RT_BUFFER_OUTPUT**) is undefined.

Flags can be used to optimize data transfers between the host and its devices. The flag `RT_BUFFER_GPU_LOCAL` can only be used in combination with `RT_BUFFER_INPUT_OUTPUT`. `RT_BUFFER_INPUT_OUTPUT` or `RT_BUFFER_GPU_LOCAL` specifies a buffer that allows the host to **only** write, and the device to read **and** write data. The written data will be never visible on the host side.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferCreate` was introduced in *OptiX* 1.0.

`RT_BUFFER_GPU_LOCAL` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromGLBO, *rtBufferDestroy*

1.11.2 rtBufferCreateFromGLBO

NAME

rtBufferCreateFromGLImage - Creates a new buffer object from an OpenGL image.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtBufferCreateFromGLBO(RTcontext context,
                                unsigned int bufferdesc,
                                unsigned int gl_id,
                                RTbuffer* buffer)
```

PARAMETERS

context

The context to create the buffer in.

bufferdesc

Bitwise or combination of the **type** and **flags** of the new buffer.

gl_id

The OpenGL image object resource handle for use in OptiX.

buffer

The return handle for the buffer object.

DESCRIPTION

rtBufferCreateFromGLBO allocates and returns a handle to a new buffer object in ***buffer** associated with **context**. Supported OpenGL buffer types are:

Pixel Buffer Objects

Vertex Buffer Objects

These buffers can be used to share data with OpenGL; changes of the content in **buffer**, either done by OpenGL or *OptiX*, will be reflected automatically in both APIs. If the size, or format, of an OpenGL buffer is changed, appropriate *OptiX* calls have to be used to update **buffer** accordingly. *OptiX* keeps only a reference to OpenGL data, when **buffer** is destroyed, the state of the **gl_id** object is unaltered.

The **type** of this buffer is specified by one of the following values in **bufferdesc**:

```
RT_BUFFER_INPUT
RT_BUFFER_OUTPUT
RT_BUFFER_INPUT_OUTPUT
```

The type values are used to specify the direction of data flow from the host to the *OptiX* devices. **RT_BUFFER_INPUT** specifies that the host may only write to the buffer and the device may only read from the buffer. **RT_BUFFER_OUTPUT** specifies the opposite, read only access on the host and write only access on the device. Devices and the host may read and write from buffers of type **RT_BUFFER_INPUT_OUTPUT**. Reading or writing to a buffer of the incorrect type (e.g., the host writing to a buffer of type **RT_BUFFER_OUTPUT**) is undefined.

Flags can be used to optimize data transfers between the host and its devices. Currently no **flags** are supported for interop buffers.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferCreateFromGLBO was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferCreate, *rtBufferDestroy*

1.11.3 rtBufferCreateFromD3D9Resource

NAME

rtBufferCreateFromD3D9Resource - Creates a new buffer object from a D3D9 resource.

SYNOPSIS

```
#include <optix_d3d9_interop.h>

RTresult RTAPI rtBufferCreateFromD3D9Resource(RTcontext context,
                                              unsigned int bufferdesc,
                                              IDirect3DResource9* resource,
                                              RTbuffer* buffer);
```

PARAMETERS

context

The context to create the buffer in.

bufferdesc

Bitwise or combination of the **type** and **flags** of the new buffer.

resource

The D3D9 resource handle for use in OptiX.

buffer

The return handle for the buffer object.

DESCRIPTION

rtBufferCreateFromD3D9Resource allocates and returns a handle to a new buffer object in ***buffer** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D9 buffer types are:

```
IDirect3DVertexBuffer9 IDirect3DIndexBuffer9
```

These buffers can be used to share data with D3D9; changes of the content in **buffer**, either done by D3D9 or *OptiX*, will be reflected automatically in both APIs. If the size, or format, of a D3D9 buffer is changed, appropriate *OptiX* calls have to be used to update **buffer** accordingly. *OptiX* keeps only a reference to D3D9 data, when **buffer** is destroyed, the state of **resource** is unaltered.

The **type** of this buffer is specified by one of the following values in **bufferdesc**:

```
RT_BUFFER_INPUT
RT_BUFFER_OUTPUT
RT_BUFFER_INPUT_OUTPUT
```

The type values are used to specify the direction of data flow from the host to the *OptiX* devices. **RT_BUFFER_INPUT** specifies that the host may only write to the buffer and the device may only read from the buffer. **RT_BUFFER_OUTPUT** specifies the opposite, read only access on the host and write only access on the device. Devices and the host may read and write from buffers of type **RT_BUFFER_INPUT_OUTPUT**. Reading or writing to a buffer of the incorrect type (e.g., the host writing to a buffer of type **RT_BUFFER_OUTPUT**) is undefined.

Flags can be used to optimize data transfers between the host and its devices. Currently no **flags** are supported for interop buffers.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferCreateFromD3D9Resource was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreate, *rtBufferDestroy*

1.11.4 rtBufferCreateFromD3D10Resource

NAME

rtBufferCreateFromD3D10Resource - Creates a new buffer object from a D3D10 resource.

SYNOPSIS

```
#include <optix_d3d10_interop.h>

RTresult RTAPI rtBufferCreateFromD3D10Resource(RTcontext context,
                                               unsigned int bufferdesc,
                                               ID3D10Resource* resource,
                                               RTbuffer* buffer);
```

PARAMETERS

context

The context to create the buffer in.

bufferdesc

Bitwise or combination of the **type** and **flags** of the new buffer.

resource

The D3D10 resource handle for use in OptiX.

buffer

The return handle for the buffer object.

DESCRIPTION

rtBufferCreateFromD3D10Resource allocates and returns a handle to a new buffer object in ***buffer** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D10 buffer types are:

ID3D10Buffer

These buffers can be used to share data with D3D10; changes of the content in **buffer**, either done by D3D10 or *OptiX*, will be reflected automatically in both APIs. If the size, or format, of a D3D10 buffer is changed, appropriate *OptiX* calls have to be used to update **buffer** accordingly. *OptiX* keeps only a reference to D3D10 data, when **buffer** is destroyed, the state of **resource** is unaltered.

The **type** of this buffer is specified by one of the following values in **bufferdesc**:

```
RT_BUFFER_INPUT
RT_BUFFER_OUTPUT
RT_BUFFER_INPUT_OUTPUT
```


The type values are used to specify the direction of data flow from the host to the *OptiX* devices. **RT_BUFFER_INPUT** specifies that the host may only write to the buffer and the device may only read from the buffer. **RT_BUFFER_OUTPUT** specifies the opposite, read only access on the host and write only access on the device. Devices and the host may read and write from buffers of type **RT_BUFFER_INPUT_OUTPUT**. Reading or writing to a buffer of the incorrect type (e.g., the host writing to a buffer of type **RT_BUFFER_OUTPUT**) is undefined.

Flags can be used to optimize data transfers between the host and its devices. Currently no **flags** are supported for interop buffers.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferCreateFromD3D10Resource was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreate, *rtBufferDestroy*

1.11.5 rtBufferCreateFromD3D11Resource

NAME

rtBufferCreateFromD3D11Resource - Creates a new buffer object from a D3D11 resource.

SYNOPSIS

```
#include <optix_d3d11_interop.h>

RTresult RTAPI rtBufferCreateFromD3D11Resource(RTcontext context,
                                               unsigned int bufferdesc,
                                               ID3D11Resource* resource,
                                               RTbuffer* buffer);
```

PARAMETERS

context

The context to create the buffer in.

bufferdesc

Bitwise or combination of the **type** and **flags** of the new buffer.

resource

The D3D11 resource handle for use in OptiX.

buffer

The return handle for the buffer object.

DESCRIPTION

rtBufferCreateFromD3D11Resource allocates and returns a handle to a new buffer object in ***buffer** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D11 buffer types are:

ID3D11Buffer

These buffers can be used to share data with D3D11; changes of the content in **buffer**, either done by D3D11 or *OptiX*, will be reflected automatically in both APIs. If the size, or format, of a D3D11 buffer is changed, appropriate *OptiX* calls have to be used to update **buffer** accordingly. *OptiX* keeps only a reference to D3D11 data, when **buffer** is destroyed, the state of **resource** is unaltered.

The **type** of this buffer is specified by one of the following values in **bufferdesc**:

```
RT_BUFFER_INPUT
RT_BUFFER_OUTPUT
RT_BUFFER_INPUT_OUTPUT
```

The type values are used to specify the direction of data flow from the host to the *OptiX* devices. **RT_BUFFER_INPUT** specifies that the host may only write to the buffer and the device may only read from the buffer. **RT_BUFFER_OUTPUT** specifies the opposite, read only access on the host and write only access on the device. Devices and the host may read and write from buffers of type **RT_BUFFER_INPUT_OUTPUT**. Reading or writing to a buffer of the incorrect type (e.g., the host writing to a buffer of type **RT_BUFFER_OUTPUT**) is undefined.

Flags can be used to optimize data transfers between the host and its devices. Currently no **flags** are supported for interop buffers.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferCreateFromD3D11Resource was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreate, *rtBufferDestroy*

1.11.6 rtBufferD3D9Unregister

NAME

rtBufferD3D9Unregister - Declares a D3D9 buffer as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D9_interop.h>

RTresult rtBufferD3D9Unregister(RTbuffer buffer)
```

PARAMETERS

buffer

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in a registered state can be unregistered via **rtBufferD3D9Register**. Once unregistered, properties like the size of the original D3D9 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with **RT_ERROR_INVALID_HANDLE**. When a buffer is already in an unregistered state **rtBufferD3D9Unregister** will return **RT_ERROR_RESOURCE_NOT_REGISTERED**.

RETURN VALUES

Relevant return values:

RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_NOT_REGISTERED

HISTORY

rtBufferD3D9Unregister was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D9Resource

1.11.7 `rtBufferD3D10Unregister`

NAME

`rtBufferD3D10Unregister` - Declares a D3D10 buffer as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D10_interop.h>

RTresult rtBufferD3D10Unregister(RTbuffer buffer)
```

PARAMETERS

`buffer`

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in a registered state can be unregistered via `rtBufferD3D10Register`. Once unregistered, properties like the size of the original D3D10 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with `RT_ERROR_INVALID_HANDLE`. When a buffer is already in an unregistered state `rtBufferD3D10Unregister` will return `RT_ERROR_RESOURCE_NOT_REGISTERED`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_NOT_REGISTERED
```

HISTORY

`rtBufferD3D10Unregister` was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D10Resource

1.11.8 rtBufferD3D11Unregister

NAME

rtBufferD3D11Unregister - Declares a D3D11 buffer as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D11_interop.h>

RTresult rtBufferD3D11Unregister(RTbuffer buffer)
```

PARAMETERS

buffer

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in a registered state can be unregistered via **rtBufferD3D11Register**. Once unregistered, properties like the size of the original D3D11 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with **RT_ERROR_INVALID_HANDLE**. When a buffer is already in an unregistered state **rtBufferD3D11Unregister** will return **RT_ERROR_RESOURCE_NOT_REGISTERED**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_RESOURCE_NOT_REGISTERED

HISTORY

rtBufferD3D11Unregister was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D11Resource

1.11.9 rtBufferD3D9Register

NAME

rtBufferD3D9Register - Declares a D3D9 buffer as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D9_interop.h>

RTresult rtBufferD3D9Register(RTbuffer buffer)
```

PARAMETERS

buffer

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in an unregistered state can be registered to OptiX again via **rtBufferD3D9Register**. Once registered, properties like the size of the original D3D9 resource cannot be modified anymore. Calls to the corresponding D3D9 functions will return with an error code. However, the data of the D3D9 resource can still be read and written by the appropriate D3D9 commands. When a buffer is already in a registered state **rtBufferD3D9Register** will return **RT_ERROR_RESOURCE_ALREADY_REGISTERED**. A resource must be registered in order to be used by OptiX. If a resource is not registered **RT_ERROR_INVALID_HANDLE** will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

rtBufferD3D9Register was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D9Resource

1.11.10 rtBufferD3D10Register

NAME

rtBufferD3D10Register - Declares a D3D10 buffer as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D10_interop.h>

Rtresult rtBufferD3D10Register(Rtbuffer buffer)
```

PARAMETERS

buffer

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in an unregistered state can be registered to OptiX again via **rtBufferD3D10Register**. Once registered, properties like the size of the original D3D10 resource cannot be modified anymore. Calls to the corresponding D3D10 functions will return with an error code. However, the data of the D3D10 resource can still be read and written by the appropriate D3D10 commands. When a buffer is already in a registered state **rtBufferD3D10Register** will return **RT_ERROR_RESOURCE_ALREADY_REGISTERED**. A resource must be registered in order to be used by OptiX. If a resource is not registered **RT_ERROR_INVALID_HANDLE** will be returned.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_RESOURCE_ALREADY_REGISTERED

HISTORY

rtBufferD3D10Register was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D10Resource

1.11.11 `rtBufferD3D11Register`

NAME

`rtBufferD3D11Register` - Declares a D3D11 buffer as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D11_interop.h>

RTresult rtBufferD3D11Register(RTbuffer buffer)
```

PARAMETERS

`buffer`

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in an unregistered state can be registered to OptiX again via `rtBufferD3D11Register`. Once registered, properties like the size of the original D3D11 resource cannot be modified anymore. Calls to the corresponding D3D11 functions will return with an error code. However, the data of the D3D11 resource can still be read and written by the appropriate D3D11 commands. When a buffer is already in a registered state `rtBufferD3D11Register` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

`rtBufferD3D11Register` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D11Resource

1.11.12 rtBufferDestroy

NAME

rtBufferDestroy - Destroys a buffer object

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtBufferDestroy(RTbuffer buffer)
```

PARAMETERS

buffer

Handle of the buffer to destroy

DESCRIPTION

rtBufferDestroy removes **buffer** from its context and deletes it. **buffer** should be a value returned by **rtBufferCreate**. After the call, **buffer** is no longer a valid handle. Any API object that referenced **buffer** will have its reference invalidated.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferDestroy was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferCreate, *rtBufferCreateFromGLBO*

1.11.13 `rtBufferGetContext`

NAME

`rtBufferGetContext` - Returns the context object that created this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetContext(RTbuffer buffer,
                           RTcontext* context)
```

PARAMETERS

`buffer`

The buffer to be queried for its context.

`context`

The return handle for the buffer's context.

DESCRIPTION

`rtBufferGetContext` returns a handle to the context that created `buffer` in `*context`. If `*context` is `NULL`, the call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate

1.11.14 rtBufferGetDimensionality

NAME

rtBufferGetDimensionality - Gets the dimensionality of this buffer object.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetDimensionality(RTbuffer buffer,
                                   unsigned int* dimensionality)
```

PARAMETERS

buffer

The buffer to be queried for its dimensionality.

dimensionality

The return handle for the buffer's dimensionality.

DESCRIPTION

rtBufferGetDimensionality returns the dimensionality of **buffer** in ***dimensionality**. The value returned will be one of 1, 2 or 3, corresponding to 1D, 2D and 3D buffers, respectively.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferGetDimensionality was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetDimensionality

1.11.15 `rtBufferGetElementSize`

NAME

`rtBufferGetElementSize` - Returns the size of a buffer's individual elements.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetElementSize(RTbuffer buffer,
                                unsigned int* element_size_return)
```

PARAMETERS

`buffer`

Specifies the buffer to be queried.

`element_size_return`

Returns the size of the buffer's individual elements.

DESCRIPTION

`rtBufferGetElementSize` queries the size of a buffer's elements. The target buffer is specified by `buffer`, which should be a value returned by `rtBufferCreate`. After the call, the size, in bytes, of the buffer's individual elements shall be returned in `*element_size_return`, if it is not NULL. Otherwise, this call has no effect.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_UNKNOWN`

HISTORY

`rtBufferGetElementSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetElementSize, *rtBufferCreate*

1.11.16 rtBufferGetFormat

NAME

rtBufferGetFormat - Gets the format of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetFormat(RTbuffer buffer,
                          RTformat* format)
```

PARAMETERS

buffer

The buffer to be queried for its format.

format

The return handle for the buffer's format.

DESCRIPTION

rtBufferGetFormat returns, in ***format**, the format of **buffer**. See **rtBufferSetFormat** for a listing of **RTbuffer** values.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferGetFormat was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetFormat, *rtBufferSetFormatUser*, *rtBufferGetFormatUser*

1.11.17 `rtBufferGetD3D9Resource`

NAME

`rtBufferGetD3D9Resource` - Gets the D3D9 resource associated with this buffer.

SYNOPSIS

```
#include <optix_d3d9_interop.h>

RTresult rtContextSetD3D9Device(RTbuffer buffer,
                                IDirect3DResource9 **resource);
```

PARAMETERS

`buffer`

The buffer to be queried for its D3D9 resource.

`resource`

The return handle for the resource.

DESCRIPTION

`rtBufferGetD3D9Resource` stores the D3D9 resource pointer in `**resource` if `buffer` was created with `rtBufferCreateFromD3D9Resource`. If `buffer` was not created from an D3D9 resource `**resource` will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtBufferGetD3D9Resource` was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D9Resource

1.11.18 rtBufferGetD3D10Resource

NAME

rtBufferGetD3D10Resource - Gets the D3D10 resource associated with this buffer.

SYNOPSIS

```
#include <optix_d3d10_interop.h>

RTresult rtContextSetD3D10Device(RTbuffer buffer,
                                ID3D10Resource **resource);
```

PARAMETERS

buffer

The buffer to be queried for its D3D10 resource.

resource

The return handle for the resource.

DESCRIPTION

rtBufferGetD3D10Resource stores the D3D10 resource pointer in ****resource** if **buffer** was created with **rtBufferCreateFromD3D10Resource**. If **buffer** was not created from an D3D10 resource ****resource** will be 0 after the call and **RT_ERROR_INVALID_VALUE** is returned.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtBufferGetD3D10Resource was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D10Resource

1.11.19 rtBufferGetD3D11Resource

NAME

rtBufferGetD3D11Resource - Gets the D3D11 resource associated with this buffer.

SYNOPSIS

```
#include <optix_d3d11_interop.h>

RTresult rtContextSetD3D11Device(RTbuffer buffer,
                                 ID3D11Resource **resource);
```

PARAMETERS

buffer

The buffer to be queried for its D3D11 resource.

resource

The return handle for the resource.

DESCRIPTION

rtBufferGetD3D11Resource stores the D3D11 resource pointer in ****resource** if **buffer** was created with **rtBufferCreateFromD3D11Resource**. If **buffer** was not created from an D3D11 resource ****resource** will be 0 after the call and **RT_ERROR_INVALID_VALUE** is returned.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtBufferGetD3D11Resource was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromD3D11Resource

1.11.20 rtBufferGetGLBOId

NAME

rtBufferGetGLBOId - Gets the OpenGL Buffer Object ID associated with this buffer.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtBufferGetGLBOId(RTbuffer buffer,
                          unsigned int *gl_id)
```

PARAMETERS

buffer

The buffer to be queried for its OpenGL buffer object id.

gl_id

The return handle for the id.

DESCRIPTION

rtBufferGetGLBOId stores the OpenGL buffer object id in **gl_id* if **buffer** was created with **rtBufferCreateFromGLBO**. If **buffer** was not created from an OpenGL Buffer Object **gl_id* will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

rtBufferGetGLBOId was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferCreateFromGLBO

1.11.21 `rtBufferGetSize1D`

NAME

`rtBufferGetSize1D` - Get the width of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetSize1D(RTbuffer buffer,
                          RTsize* width)
```

PARAMETERS

`buffer`

The buffer to be queried for its dimensions.

`width`

The return handle for the buffer's width.

DESCRIPTION

`rtBufferGetSize1D` stores the width of `buffer` in `*width`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferGetSize1D` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize1D, *rtBufferSetSize2D*, *rtBufferSetSize3D*, *rtBufferSetSizev*, *rtBufferGetSize2D*, *rtBufferGetSize3D*, *rtBufferGetSizev*

1.11.22 rtBufferGetSize2D

NAME

rtBufferGetSize2D - Gets the width and height of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetSize2D(RTbuffer buffer,
                           RTsize* width,
                           RTsize* height)
```

PARAMETERS

buffer

The buffer to be queried for its dimensions.

width

The return handle for the buffer's width.

height

The return handle for the buffer's height.

DESCRIPTION

rtBufferGetSize2D stores the width and height of **buffer** in ***width** and ***height**, respectively.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferGetSize2D was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize1D, *rtBufferSetSize2D*, *rtBufferSetSize3D*, *rtBufferSetSizev*, *rtBufferGetSize1D*, *rtBufferGetSize3D*, *rtBufferGetSizev*

1.11.23 rtBufferGetSize3D

NAME

rtBufferGetSize3D - Gets the width, height and depth of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetSize3D(RTbuffer buffer,
                           RTsize* width,
                           RTsize* height,
                           RTsize* depth)
```

PARAMETERS

buffer

The buffer to be queried for its dimensions.

width

The return handle for the buffer's width.

height

The return handle for the buffer's height.

depth

The return handle for the buffer's depth.

DESCRIPTION

rtBufferGetSize3D stores the width, height and depth of **buffer** in ***width**, ***height** and ***depth**, respectively.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtBufferGetSize3D` was introduced in *OptiX* 1.0.

SEE ALSO

`rtBufferSetSize1D`, `rtBufferSetSize2D`, `rtBufferSetSize3D`, `rtBufferSetSizev`, `rtBufferGetSize1D`, `rtBufferGetSize2D`, `rtBufferGetSizev`

1.11.24 rtBufferGetSizev

NAME

rtBufferGetSizev - Gets the dimensions of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferGetSizev(RTbuffer buffer,
                          unsigned int dimensionality,
                          RTsize* dims)
```

PARAMETERS

buffer

The buffer to be queried for its dimensions.

dimensionality

The number of requested dimensions.

dims

The array of dimensions the call will store to.

DESCRIPTION

rtBufferGetSizev stores the dimensions of **buffer** in ***dims**. The number of dimensions returned is specified by **dimensionality**. The storage at **dims** must be large enough to hold the number of requested buffer dimensions.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferGetSizev was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferGetDimensionality

1.11.25 rtBufferGLUnregister

NAME

rtBufferGLUnregister - Declares an OpenGL buffer as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtBufferGLUnregister(RTbuffer buffer)
```

PARAMETERS

buffer

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in a registered state can be unregistered via **rtBufferGLRegister**. Once unregistered, properties like the size of the original GL resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with **RT_ERROR_INVALID_HANDLE**. When a buffer is already in an unregistered state **rtBufferGLUnregister** will return **RT_ERROR_RESOURCE_NOT_REGISTERED**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_RESOURCE_NOT_REGISTERED

HISTORY

rtBufferGLUnregister was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromGLBO

1.11.26 `rtBufferGLRegister`

NAME

`rtBufferGLRegister` - Declares an OpenGL buffer as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_gl_interop.h>

Rtresult rtBufferGLRegister(Rtbuffer buffer)
```

PARAMETERS

`buffer`

The handle for the buffer object.

DESCRIPTION

An OptiX buffer in an unregistered state can be registered to OptiX again via `rtBufferGLRegister`. Once registered, properties like the size of the original GL resource cannot be modified anymore. Calls to the corresponding GL functions will return with an error code. However, the data of the GL resource can still be read and written by the appropriate GL commands. When a buffer is already in a registered state `rtBufferGLRegister` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_RESOURCE_ALREADY_REGISTERED`

HISTORY

`rtBufferGLRegister` was introduced in *OptiX 2.0*.

SEE ALSO

rtBufferCreateFromGLBO

1.11.27 rtBufferMap

NAME

rtBufferMap - Maps a buffer object to the host.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferMap(RTbuffer buffer,
                    void** user_pointer)
```

PARAMETERS

buffer

The buffer to be mapped.

user_pointer

Return handle to a user pointer where the buffer will be mapped to.

DESCRIPTION

rtBufferMap returns a pointer, accessible by the host, in ***user_pointer** that contains a mapped copy of the contents of **buffer**. The memory pointed to by ***user_pointer** can be written to or read from, depending on the type of **buffer**. For example, this code snippet demonstrates creating and filling an input buffer with floats.

```
RTbuffer buffer;
float* data;
rtBufferCreate(context, RT_BUFFER_INPUT, &buffer);
rtBufferSetFormat(buffer, RT_FORMAT_FLOAT);
rtBufferSetSize1D(buffer, 10);
rtBufferMap(buffer, (void*)&data);
for(int i = 0; i < 10; ++i)
    data[i] = 4.f * i;
rtBufferUnmap(buffer);
```

If **buffer** has already been mapped, the call will return **RT_ERROR_ALREADY_MAPPED**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_ALREADY_MAPPED
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferMap was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferUnmap

1.11.28 rtBufferSetElementSize

NAME

rtBufferSetElementSize - Modifies the size in bytes of a buffer's individual elements.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferSetElementSize(RTbuffer buffer,
                                unsigned int element_size)
```

PARAMETERS

buffer

Specifies the buffer to be modified.

element_size

Specifies the new size in bytes of the buffer's individual elements.

DESCRIPTION

rtBufferSetElementSize modifies the size in bytes of a buffer's user-formatted elements. The target buffer is specified by **buffer**, which should be a value returned by **rtBufferCreate** and should have format **RT_FORMAT_USER**. The new size of the buffer's individual elements is specified by **element_size** and should be a value not equal to 0. If the buffer has format **RT_FORMAT_USER**, and **element_size** is not equal to 0, then after the call, the buffer's individual elements shall have size equal to **element_size** and all storage associated with the buffer shall be reset. Otherwise, this call has no effect and returns either **RT_TYPE_MISMATCH** if the buffer does not have format **RT_FORMAT_USER** or **RT_INVALID_VALUE** if the buffer has format **RT_FORMAT_USER** but **element_size** is equal to 0.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_TYPE_MISMATCH

RT_ERROR_INVALID_VALUE

RT_ERROR_INVALID_CONTEXT

HISTORY

`rtBufferSetElementSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferGetElementSize, *rtBufferCreate*

1.11.29 rtBufferSetSize1D

NAME

rtBufferSetSize1D - Sets the width and dimensionality of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferSetSize1D(RTbuffer buffer,
                          RTsize width)
```

PARAMETERS

buffer

The buffer to be resized.

width

The width of the resized buffer.

DESCRIPTION

rtBufferSetSize1D sets the dimensionality of **buffer** to 1 as well as setting its width to **width**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferSetSize1D was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize2D, *rtBufferSetSize3D*, *rtBufferSetSizev*, *rtBufferGetSize1D*, *rtBufferGetSize2D*, *rtBufferGetSize3D*, *rtBufferGetSizev*

1.11.30 `rtBufferSetSize2D`

NAME

`rtBufferSetSize2D` - Sets the width, height and dimensionality of this buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferSetSize2D(RTbuffer buffer,
                           RTsize width,
                           RTsize height)
```

PARAMETERS

buffer

The buffer to be resized.

width

The width of the resized buffer.

height

The height of the resized buffer.

DESCRIPTION

`rtBufferSetSize2D` sets the dimensionality of **buffer** to 2 as well as setting its width and height to **width** and **height**, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferSetSize2D` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize1D, rtBufferSetSize3D, rtBufferSetSizev, rtBufferGetSize1D, rtBufferGetSize2D, rtBufferGetSize3D, rtBufferGetSizev

1.11.31 `rtBufferSetSize3D`

NAME

`rtBufferSetSize3D` - Sets the width, height, depth and dimensionality of a buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferSetSize3D(RTbuffer buffer,
                          RTsize width,
                          RTsize height,
                          RTsize depth)
```

PARAMETERS

buffer

The buffer to be resized.

width

The width of the resized buffer.

height

The height of the resized buffer.

depth

The depth of the resized buffer.

DESCRIPTION

`rtBufferSetSize3D` sets the dimensionality of **buffer** to 3 as well as setting its width, height and depth to **width**, **height** and **depth**, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferSetSize3D` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize2D, *rtBufferSetSize3D*, *rtBufferSetSizev*, *rtBufferGetSize1D*, *rtBufferGetSize2D*, *rtBufferGetSize3D*, *rtBufferGetSizev*

1.11.32 `rtBufferSetSizev`

NAME

`rtBufferSetSizev` - Sets the dimensionality and dimensions of a buffer.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferSetSizev(RTbuffer buffer,
                          unsigned int dimensionality,
                          const RTsize* dims)
```

PARAMETERS

buffer

The buffer to be resized.

dimensionality

The dimensionality the buffer will be resized to.

dims

The array of sizes for the dimension of the resize.

DESCRIPTION

`rtBufferSetSizev` sets the dimensionality of **buffer** to **dimensionality** as well as setting the dimensions of the buffer to the values stored at ***dims**, which must contain a number of values equal to **dimensionality**.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferSetSizev` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferSetSize1D, rtBufferSetSize2D, rtBufferSetSize3D, rtBufferGetSize1D, rtBufferGetSize2D, rtBufferGetSize3D, rtBufferGetSizev

1.11.33 `rtBufferUnmap`

NAME

`rtBufferUnmap` - Unmaps a buffer's storage from the host.

SYNOPSIS

```
#include <optix.h>

RTresult rtBufferUnmap(RTbuffer buffer)
```

PARAMETERS

`buffer`

The buffer to unmap.

DESCRIPTION

`rtBufferUnmap` unmaps a buffer from the host after a call to `rtBufferMap`. `rtContextLaunch*` cannot be called while buffers are still mapped to the host. A call to `rtBufferUnmap` that does not follow a matching `rtBufferMap` call will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtBufferUnmap` was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferMap

1.11.34 rtBufferValidate

NAME

rtBufferValidate - Validates the state of a buffer.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtBufferValidate(RTbuffer buffer)
```

PARAMETERS

buffer

The buffer to validate.

DESCRIPTION

rtBufferValidate checks **buffer** for completeness. If **buffer** has not had its dimensionality, size or format set, this call will return **RT_ERROR_INVALID_CONTEXT**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtBufferValidate was introduced in *OptiX* 1.0.

SEE ALSO

rtBufferCreate, *rtBufferCreateFromGLBO* *rtContextValidate*

1.12 Texture Sampler

NAME

Texture Sampler

DESCRIPTION

This section describes the API functions for creation and handling of TextureSampler objects.

`rtTextureSamplerCreate`

`rtTextureSamplerCreateFromD3D9Resource`

`rtTextureSamplerCreateFromD3D10Resource`

`rtTextureSamplerCreateFromD3D11Resource`

`rtTextureSamplerD3D9Unregister`

`rtTextureSamplerD3D10Unregister`

`rtTextureSamplerD3D11Unregister`

`rtTextureSamplerD3D9Register`

`rtTextureSamplerD3D10Register`

`rtTextureSamplerD3D11Register`

`rtTextureSamplerDestroy`

`rtTextureSamplerGetArraySize`

`rtTextureSamplerGetBuffer`

`rtTextureSamplerGetContext`

`rtTextureSamplerGetD3D9Resource`

`rtTextureSamplerGetD3D10Resource`

`rtTextureSamplerGetD3D11Resource`

`rtTextureSamplerGetFilteringModes`

`rtTextureSamplerGetGLImageId`

`rtTextureSamplerGetIndexingMode`

`rtTextureSamplerGetMaxAnisotropy`

`rtTextureSamplerGetMipLevelCount`

`rtTextureSamplerGetReadMode`

`rtTextureSamplerGetWrapMode`

`rtTextureSamplerGLUnregister`

`rtTextureSamplerGLRegister`
`rtTextureSamplerSetArraySize`
`rtTextureSamplerSetBuffer`
`rtTextureSamplerSetFilteringModes`
`rtTextureSamplerSetIndexingMode`
`rtTextureSamplerSetMaxAnisotropy`
`rtTextureSamplerSetMipLevelCount`
`rtTextureSamplerSetReadMode`
`rtTextureSamplerSetWrapMode`
`rtTextureSamplerValidate`

HISTORY

TextureSample objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Buffer, Variables, Context-Free Functions

1.12.1 `rtTextureSamplerCreate`

NAME

`rtTextureSamplerCreate` - Creates a new texture sampler object.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerCreate(RTcontext context,
                               RTtexturesampler* texturesampler)
```

PARAMETERS

`context`

The context the texture sampler object will be created in.

`texturesampler`

The return handle to the new texture sampler object.

DESCRIPTION

`rtTextureSamplerCreate` allocates and returns a new handle to a texture sampler object, in `*texturesampler`, and associates it with `context`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerCreate` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerDestroy

1.12.2 rtTextureSamplerCreateFromGLImage

NAME

rtTextureSamplerCreateFromGLImage - Creates a new texture sampler object from an OpenGL image.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtTextureSamplerCreateFromGLImage(RTcontext context,
                                           unsigned int gl_id,
                                           RTgltarget target,
                                           RTtexturesampler* texturesampler)
```

PARAMETERS

context

The context to create the buffer in.

gl_id

The OpenGL image object resource handle for use in OptiX.

target

The OpenGL target.

texturesampler

The return handle for the texture sampler object.

DESCRIPTION

rtTextureSamplerCreateFromGLImage allocates and returns a handle to a new texture sampler object in **texturesampler* associated with **context**. If the allocated size of the GL texture is 0, `RT_ERROR_MEMORY_ALLOCATION_FAILED` will be returned. Supported OpenGL image types are:

Renderbuffers

`GL_TEXTURE_2D`

`GL_TEXTURE_2D_RECT`

`GL_TEXTURE_3D`

These types are reflected by **target**:

`RT_TARGET_GL_RENDER_BUFFER`

`RT_TARGET_GL_TEXTURE_2D`

`RT_TARGET_GL_TEXTURE_RECTANGLE`

RT_TARGET_GL_TEXTURE_3D

Supported attachment points for renderbuffers are:

GL_COLOR_ATTACHMENT<NUM>

These texture samplers can be used to share data with OpenGL; changes of the content and size of **texturesampler** done by OpenGL will be reflected automatically in OptiX. Currently texture sampler data are read only in *OptiX* programs. *OptiX* keeps only a reference to OpenGL data, when **texturesampler** is destroyed, the state of the **glid** image is unaltered.

The array size and number of mipmap levels can't be changed for texture samplers that encapsulate a GL image. Furthermore no buffer objects can be queried.

Currently OptiX supports only a limited number of internal OpenGL texture formats. Texture formats with an internal type of float, e.g. GL_RGBA32F, and many integer formats are supported. Depth formats as well as multisample buffers are also currently not supported. Please refer to the *Appendix* for a complete list of supported texture formats.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtTextureSamplerCreateFromGLImage was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreate, *rtTextureSamplerDestroy*

1.12.3 rtTextureSamplerCreateFromD3D9Resource

NAME

rtTextureSamplerCreateFromD3D9Resource - Creates a new texture sampler object from an D3D9 resource.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtTextureSamplerCreateFromD3D9Resource(RTcontext context,
                                                IDirect3DResource9* resource,
                                                RTtexturesampler* texturesampler);
```

PARAMETERS

context

The context to create the buffer in.

resource

The D3D9 resource handle for use in OptiX.

texturesampler

The return handle for the texture sampler object.

DESCRIPTION

rtTextureSamplerCreateFromD3D9Resource allocates and returns a handle to a new texture sampler object in ***texturesampler** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D9 texture types are:

IDirect3DSurface9

(derivatives of) IDirect3DBaseTexture9

These texture samplers can be used to share data with D3D9; changes of the content and size of **texture-sampler** done by D3D9 will be reflected automatically in OptiX. Currently texture sampler data are read only in *OptiX* programs. *OptiX* keeps only a reference to D3D9 data, when **texturesampler** is destroyed, the state of the **resource** is unaltered.

The array size and number of mipmap levels can't be changed for texture samplers that encapsulate a D3D9 resource. Furthermore no buffer objects can be queried. Please refer to the *Appendix* for a complete list of supported texture formats.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTextureSamplerCreateFromD3D9Resource` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreate, *rtTextureSamplerDestroy*

1.12.4 rtTextureSamplerCreateFromD3D10Resource

NAME

rtTextureSamplerCreateFromD3D10Resource - Creates a new texture sampler object from an D3D10 resource.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtTextureSamplerCreateFromD3D10Resource(RTcontext context,
                                                ID3D10Resource* resource,
                                                RTtexturesampler* texturesampler);
```

PARAMETERS

context

The context to create the buffer in.

resource

The D3D10 resource handle for use in OptiX.

texturesampler

The return handle for the texture sampler object.

DESCRIPTION

rtTextureSamplerCreateFromD3D10Resource allocates and returns a handle to a new texture sampler object in ***texturesampler** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D10 texture types are:

ID3D10Texture1D

ID3D10Texture2D

ID3D10Texture3D

These texture samplers can be used to share data with D3D10; changes of the content and size of **texture-sampler** done by D3D10 will be reflected automatically in OptiX. Currently texture sampler data are read only in *OptiX* programs. *OptiX* keeps only a reference to D3D10 data, when **texturesampler** is destroyed, the state of the **resource** is unaltered.

The array size and number of mipmap levels can't be changed for texture samplers that encapsulate a D3D10 resource. Furthermore no buffer objects can be queried. Please refer to the *Appendix* for a complete list of supported texture formats.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTextureSamplerCreateFromD3D10Resource` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreate, *rtTextureSamplerDestroy*

1.12.5 rtTextureSamplerCreateFromD3D11Resource

NAME

rtTextureSamplerCreateFromD3D11Resource - Creates a new texture sampler object from an D3D11 resource.

SYNOPSIS

```
#include <optix_gl_interop.h>
```

```
RTresult rtTextureSamplerCreateFromD3D11Resource(RTcontext context,
                                                ID3D11Resource* resource,
                                                RTtexturesampler* texturesampler);
```

PARAMETERS

context

The context to create the buffer in.

resource

The D3D11 resource handle for use in OptiX.

texturesampler

The return handle for the texture sampler object.

DESCRIPTION

rtTextureSamplerCreateFromD3D11Resource allocates and returns a handle to a new texture sampler object in ***texturesampler** associated with **context**. If the allocated size of the D3D resource is 0, **RT_ERROR_MEMORY_ALLOCATION_FAILED** will be returned. Supported D3D11 texture types are:

ID3D11Texture1D

ID3D11Texture2D

ID3D11Texture3D

These texture samplers can be used to share data with D3D11; changes of the content and size of **texture-sampler** done by D3D11 will be reflected automatically in OptiX. Currently texture sampler data are read only in *OptiX* programs. *OptiX* keeps only a reference to D3D11 data, when **texturesampler** is destroyed, the state of the **resource** is unaltered.

The array size and number of mipmap levels can't be changed for texture samplers that encapsulate a D3D11 resource. Furthermore no buffer objects can be queried. Please refer to the *Appendix* for a complete list of supported texture formats.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTextureSamplerCreateFromD3D11Resource` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreate, *rtTextureSamplerDestroy*

1.12.6 `rtTextureSamplerD3D9Unregister`

NAME

`rtTextureSamplerD3D9Unregister` - Declares a D3D9 texture as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D9_interop.h>

RTresult rtTextureSamplerD3D9Unregister(RTtexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture sampler object.

DESCRIPTION

An OptiX texture sampler in a registered state can be unregistered via `rtTextureSamplerD3D9Unregister`. Once unregistered, properties like the size of the original D3D9 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with `RT_ERROR_INVALID_HANDLE`. When a buffer is already in an unregistered state `rtBufferD3D9Unregister` will return `RT_ERROR_RESOURCE_NOT_REGISTERED`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_NOT_REGISTERED
```

HISTORY

`rtTextureSamplerD3D9Unregister` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreateFromD3D9Resource

1.12.7 `rtTextureSamplerD3D10Unregister`

NAME

`rtTextureSamplerD3D10Unregister` - Declares a D3D10 texture as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D10_interop.h>

RTresult rtTextureSamplerD3D10Unregister(RTtexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture sampler object.

DESCRIPTION

An OptiX texture sampler in a registered state can be unregistered via `rtTextureSamplerD3D10Unregister`. Once unregistered, properties like the size of the original D3D10 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with `RT_ERROR_INVALID_HANDLE`. When a buffer is already in an unregistered state `rtBufferD3D10Unregister` will return `RT_ERROR_RESOURCE_NOT_REGISTERED`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_NOT_REGISTERED
```

HISTORY

`rtTextureSamplerD3D10Unregister` was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreateFromD3D9Resource

1.12.8 `rtTextureSamplerD3D11Unregister`

NAME

`rtTextureSamplerD3D11Unregister` - Declares a D3D11 texture as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_D3D11_interop.h>
```

```
RTresult rtTextureSamplerD3D11Unregister(RTtexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture sampler object.

DESCRIPTION

An OptiX texture sampler in a registered state can be unregistered via `rtTextureSamplerD3D11Unregister`. Once unregistered, properties like the size of the original D3D11 resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with `RT_ERROR_INVALID_HANDLE`. When a buffer is already in an unregistered state `rtBufferD3D11Unregister` will return `RT_ERROR_RESOURCE_NOT_REGISTERED`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_RESOURCE_NOT_REGISTERED`

HISTORY

`rtTextureSamplerD3D11Unregister` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreateFromD3D11Resource

1.12.9 `rtTextureSamplerD3D9Register`

NAME

`rtTextureSamplerD3D9Register` - Declares an D3D9 texture as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D9_interop.h>

Rtresult rtTextureSamplerD3D9Register(Rttexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture object.

DESCRIPTION

An OptiX texture sampler in an unregistered state can be registered to OptiX again via `rtTextureSamplerD3D9Register`. Once registered, properties like the size of the original D3D9 resource cannot be modified anymore. Calls to the corresponding D3D9 functions will return with an error code. However, the data of the D3D9 resource can still be read and written by the appropriate D3D9 commands. When a texture sampler is already in a registered state `rtTextureSamplerD3D9Register` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

`rtTextureSamplerD3D9Register` was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreateFromD3D9Resource

1.12.10 `rtTextureSamplerD3D10Register`

NAME

`rtTextureSamplerD3D10Register` - Declares an D3D10 texture as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D10_interop.h>

RTresult rtTextureSamplerD3D10Register(RTtexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture object.

DESCRIPTION

An OptiX texture sampler in an unregistered state can be registered to OptiX again via `rtTextureSamplerD3D10Register`. Once registered, properties like the size of the original D3D10 resource cannot be modified anymore. Calls to the corresponding D3D10 functions will return with an error code. However, the data of the D3D10 resource can still be read and written by the appropriate D3D10 commands. When a texture sampler is already in a registered state `rtTextureSamplerD3D10Register` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

`rtTextureSamplerD3D10Register` was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreateFromD3D10Resource

1.12.11 `rtTextureSamplerD3D11Register`

NAME

`rtTextureSamplerD3D11Register` - Declares an D3D11 texture as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_D3D11_interop.h>

Rtresult rtTextureSamplerD3D11Register(Rttexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture object.

DESCRIPTION

An OptiX texture sampler in an unregistered state can be registered to OptiX again via `rtTextureSamplerD3D11Register`. Once registered, properties like the size of the original D3D11 resource cannot be modified anymore. Calls to the corresponding D3D11 functions will return with an error code. However, the data of the D3D11 resource can still be read and written by the appropriate D3D11 commands. When a texture sampler is already in a registered state `rtTextureSamplerD3D11Register` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

`rtTextureSamplerD3D11Register` was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreateFromD3D11Resource

1.12.12 `rtTextureSamplerDestroy`

NAME

`rtTextureSamplerDestroy` - Destroys a texture sampler object

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtTextureSamplerDestroy(RTtexturesampler texturesampler)
```

PARAMETERS

`texturesampler`

Handle of the texture sampler to destroy

DESCRIPTION

`rtTextureSamplerDestroy` removes `texturesampler` from its context and deletes it. `texturesampler` should be a value returned by `rtTextureSamplerCreate`. After the call, `texturesampler` is no longer a valid handle. Any API object that referenced `texturesampler` will have its reference invalidated.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerDestroy` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerCreate

1.12.13 `rtTextureSamplerGetArraySize`

NAME

`rtTextureSamplerGetArraySize` - Gets the number of array slices present in a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetArraySize(RTtexturesampler texturesampler,
                                     unsigned int* num_textures_in_array)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`num_textures_in_array`

The return handle for the number of texture slices the texture sampler.

DESCRIPTION

`rtTextureSamplerGetArraySize` gets the number of texture array slices in `texturesampler` and stores it in `*num_textures_in_array`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetArraySize` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetArraySize

1.12.14 `rtTextureSamplerGetBuffer`

NAME

`rtTextureSamplerGetBuffer` - Gets a buffer object handle from a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetBuffer(RTtexturesampler texturesampler,
                                   unsigned int texture_array_idx,
                                   unsigned int mip_level,
                                   RTbuffer* buffer)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried for the buffer.

`texture_array_idx`

The array slice index the buffer will be queried from.

`mip_level`

The MIP level the buffer will be queried from.

`buffer`

The return handle to the buffer attached to the texture sampler.

DESCRIPTION

`rtTextureSamplerGetBuffer` gets a buffer object from `texturesampler` from the specified MIP level and array slice and stores it in `*buffer`. `mip_level` and `texture_array_idx` specify the MIP level and array slice, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetBuffer` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetBuffer

1.12.15 rtTextureSamplerGetContext

NAME

rtTextureSamplerGetContext - Gets the context object that created this texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetContext(RTtexturesampler texturesampler,
                                    RTcontext* context)
```

PARAMETERS

texturesampler

The texture sampler object to be queried for its context.

context

The return handle for the context object of the texture sampler.

DESCRIPTION

rtTextureSamplerGetContext returns a handle to the context object that was used to create **texture-sampler**. If **context** is NULL, the call will return **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

rtTextureSamplerGetContext was introduced in *OptiX* 1.0.

SEE ALSO

rtContextCreate

1.12.16 `rtTextureSamplerGetD3D9Resource`

NAME

`rtTextureSamplerGetD3D9Resource` - Gets the D3D9 resource associated with this texture sampler.

SYNOPSIS

```
#include <optix_d3d9_interop.h>

RTresult rtTextureSamplerGetD3D9Resource(RTtexturesampler sampler,
                                         Direct3DResource9 **resource);
```

PARAMETERS

`sampler`

The texture sampler to be queried for its D3D9 resource.

`resource`

The return handle for the resource.

DESCRIPTION

`rtTextureSamplerGetD3D9Resource` stores the D3D9 resource pointer in ****resource** if **sampler** was created with `rtTextureSamplerGetD3D9Resource`. If **sampler** was not created from an D3D9 resource **resource** will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtTextureSamplerGetD3D9Resource` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D9Resource

1.12.17 `rtTextureSamplerGetD3D10Resource`

NAME

`rtTextureSamplerGetD3D10Resource` - Gets the D3D10 resource associated with this texture sampler.

SYNOPSIS

```
#include <optix_d3d10_interop.h>

RTresult rtTextureSamplerGetD3D10Resource(RTtexturesampler sampler,
                                          D3D10Resource **resource);
```

PARAMETERS

`sampler`

The texture sampler to be queried for its D3D10 resource.

`resource`

The return handle for the resource.

DESCRIPTION

`rtTextureSamplerGetD3D10Resource` stores the D3D10 resource pointer in `**resource` if `sampler` was created with `rtTextureSamplerGetD3D10Resource`. If `sampler` was not created from an D3D10 resource `resource` will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtTextureSamplerGetD3D10Resource` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D10Resource

1.12.18 `rtTextureSamplerGetD3D11Resource`

NAME

`rtTextureSamplerGetD3D11Resource` - Gets the D3D11 resource associated with this texture sampler.

SYNOPSIS

```
#include <optix_d3d11_interop.h>

RTresult rtTextureSamplerGetD3D11Resource(RTtexturesampler sampler,
                                          D3D11Resource **resource);
```

PARAMETERS

`sampler`

The texture sampler to be queried for its D3D11 resource.

`resource`

The return handle for the resource.

DESCRIPTION

`rtTextureSamplerGetD3D11Resource` stores the D3D11 resource pointer in `**resource` if `sampler` was created with `rtTextureSamplerGetD3D11Resource`. If `sampler` was not created from an D3D11 resource `resource` will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtTextureSamplerGetD3D11Resource` was introduced in *OptiX* 2.0.

SEE ALSO

rtBufferCreateFromD3D11Resource

1.12.19 `rtTextureSamplerGetFilteringModes`

NAME

`rtTextureSamplerGetFilteringModes` - Gets the filtering modes of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetFilteringModes(RTtexturesampler texturesampler,
                                           RTfiltermode* minification,
                                           RTfiltermode* magnification,
                                           RTfiltermode* mipmapping)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`minification`

The return handle for the minification filtering mode of the texture sampler.

`magnification`

The return handle for the magnification filtering mode of the texture sampler.

`mipmapping`

The return handle for the MIP mapping filtering mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerGetFilteringModes` gets the minification, magnification and MIP mapping filtering modes from `texturesampler` and stores them in `*minification`, `*magnification` and `*mipmapping`, respectively. See `rtTextureSamplerSetFilteringModes` for the values `RTfiltermode` may take.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetFilteringModes` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetFilteringModes

1.12.20 `rtTextureSamplerGetGLImageId`

NAME

`rtTextureSamplerGetGLImageId` - Gets the OpenGL image object id associated with this texture sampler.

SYNOPSIS

```
#include <optix_gl_interop.h>
```

```
RTresult rtTextureSamplerGetGLImageId(RTtexturesampler sampler,  
                                       unsigned int *gl_id)
```

PARAMETERS

`sampler`

The texture sampler to be queried for its OpenGL image object id.

`gl_id`

The return handle for the id.

DESCRIPTION

`rtTextureSamplerGetGLImageId` stores the OpenGL image object id in `*gl_id` if `sampler` was created with `rtTextureSamplerGetGLImageId`. If `sampler` was not created from an OpenGL image object `gl_id` will be 0 after the call and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetGLImageId` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreateFromGLImage

1.12.21 `rtTextureSamplerGetIndexingMode`

NAME

`rtTextureSamplerGetIndexingMode` - Gets the indexing mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetIndexingMode(RTtexturesampler texturesampler,
                                         RTtextureindexmode* indexmode)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`indexmode`

The return handle for the indexing mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerGetIndexingMode` gets the indexing mode of `texturesampler` and stores it in `*indexmode`. See `rtTextureSamplerSetIndexingMode` for the values `RTtextureindexmode` may take.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetIndexingMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetIndexingMode

1.12.22 `rtTextureSamplerGetMaxAnisotropy`

NAME

`rtTextureSamplerGetMaxAnisotropy` - Gets the maximum anisotropy level for a texture sampler.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtTextureSamplerGetMaxAnisotropy(RTtexturesampler texturesampler,  
                                          float* value)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`wrapmode`

The return handle for the maximum anisotropy level of the texture sampler.

DESCRIPTION

`rtTextureSamplerGetMaxAnisotropy` gets the maximum anisotropy level for `texturesampler` and stores it in `*value`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetMaxAnisotropy` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetMaxAnisotropy

1.12.23 `rtTextureSamplerGetMipLevelCount`

NAME

`rtTextureSamplerGetMipLevelCount` - Gets the number of MIP levels in a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetMipLevelCount(RTtexturesampler texturesampler,
                                          unsigned int* num_mip_levels)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`num_mip_levels`

The return handle for the number of MIP levels in the texture sampler.

DESCRIPTION

`rtTextureSamplerGetMipLevelCount` gets the number of MIP levels contained in `texturesampler` and stores it in `*num_mip_levels`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetMipLevelCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetMipLevelCount

1.12.24 `rtTextureSamplerGetReadMode`

NAME

`rtTextureSamplerGetReadMode` - Gets the read mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetReadMode(RTtexturesampler texturesampler,
                                     RTtexturereadmode* readmode)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`readmode`

The return handle for the read mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerGetReadMode` gets the read mode of `texturesampler` and stores it in `*readmode`. See `rtTextureSamplerSetReadMode` for a list of values `RTtexturereadmode` can take.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetReadMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetReadMode

1.12.25 `rtTextureSamplerGetWrapMode`

NAME

`rtTextureSamplerGetWrapMode` - Gets the wrap mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerGetWrapMode(RTtexturesampler texturesampler,
                                     RTwrapmode* wrapmode)
```

PARAMETERS

`texturesampler`

The texture sampler object to be queried.

`wrapmode`

The return handle for the wrap mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerGetWrapMode` gets the texture wrapping mode of `texturesampler` and stores it in `*wrapmode`. See `rtTextureSamplerSetWrapMode` for a list of values `RTwrapmode` can take.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerGetWrapMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerSetWrapMode

1.12.26 `rtTextureSamplerGLUnregister`

NAME

`rtTextureSamplerGLUnregister` - Declares a OpenGL texture as mutable and inaccessible by OptiX.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtTextureSamplerGLUnregister(RTtexturesampler sampler)
```

PARAMETERS

sampler

The handle for the texture sampler object.

DESCRIPTION

An OptiX texture sampler in a registered state can be unregistered via `rtTextureSamplerGLUnregister`. Once unregistered, properties like the size of the original GL resource can be changed. As long as a resource is unregistered, OptiX will not be able to access the data and will fail with `RT_ERROR_INVALID_HANDLE`. When a buffer is already in an unregistered state `rtBufferGLUnregister` will return `RT_ERROR_RESOURCE_NOT_REGISTERED`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_NOT_REGISTERED
```

HISTORY

`rtTextureSamplerGLUnregister` was introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreateFromGLImage

1.12.27 `rtTextureSamplerGLRegister`

NAME

`rtTextureSamplerGLRegister` - Declares an OpenGL texture as immutable and accessible by OptiX.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtTextureSamplerGLRegister(RTtexturesampler sampler)
```

PARAMETERS

`sampler`

The handle for the texture object.

DESCRIPTION

An OptiX texture sampler in an unregistered state can be registered to OptiX again via `rtTextureSamplerGLRegister`. Once registered, properties like the size of the original GL resource cannot be modified anymore. Calls to the corresponding GL functions will return with an error code. However, the data of the GL resource can still be read and written by the appropriate GL commands. When a texture sampler is already in a registered state `rtTextureSamplerGLRegister` will return `RT_ERROR_RESOURCE_ALREADY_REGISTERED`. A resource must be registered in order to be used by OptiX. If a resource is not registered `RT_ERROR_INVALID_HANDLE` will be returned.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_RESOURCE_ALREADY_REGISTERED
```

HISTORY

`rtTextureSamplerGLRegister` was introduced in *OptiX* 2.0.

SEE ALSO

rtTextureSamplerCreateFromGLImage

1.12.28 `rtTextureSamplerSetArraySize`

NAME

`rtTextureSamplerSetArraySize` - Sets the array size of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetArraySize(RTtexturesampler texturesampler,
                                     unsigned int  num_textures_in_array)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`wrapmode`

The new number of array slices of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetArraySize` specifies the number of texture array slices present in `texturesampler` as `num_textures_in_array`. After changing the number of slices in the array, buffers must be reassociated with `texturesampler` via `rtTextureSamplerSetBuffer`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerSetArraySize` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetArraySize

1.12.29 `rtTextureSamplerSetBuffer`

NAME

`rtTextureSamplerSetBuffer` - Attaches a buffer object to a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetBuffer(RTtexturesampler texturesampler,
                                   unsigned int texture_array_idx,
                                   unsigned int mip_level,
                                   RTbuffer buffer)
```

PARAMETERS

`texturesampler`

The texture sampler object that will contain the buffer.

`texture_array_idx`

The array slice index the buffer will be attached to.

`mip_level`

The MIP level the buffer will be attached to.

`buffer`

The buffer to be attached to the texture sampler.

DESCRIPTION

`rtTextureSamplerSetBuffer` attaches `buffer` to `texturesampler` at the specified array slice and MIP level. The array slice and MIP level are specified by `texture_array_idx` and `mip_level`, respectively.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerSetBuffer` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetBuffer

1.12.30 `rtTextureSamplerSetFilteringModes`

NAME

`rtTextureSamplerSetFilteringModes` - Sets the filtering modes of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetFilteringModes(RTtexturesampler texturesampler,
                                           RTfiltermode minification,
                                           RTfiltermode magnification,
                                           RTfiltermode mipmapping)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`minification`

The new minification filter mode of the texture sampler.

`magnification`

The new magnification filter mode of the texture sampler.

`mipmapping`

The new MIP mapping filter mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetFilteringModes` sets the minification, magnification and MIP mapping filter modes for `texturesampler`. `RTfiltermode` must be one of the following values:

```
RT_FILTER_NEAREST
RT_FILTER_LINEAR
RT_FILTER_NONE
```

These filter modes specify how the texture sampler will interpolate buffer data that has been attached to it. `minification` and `magnification` must be one of `RT_FILTER_NEAREST` or `RT_FILTER_LINEAR`. `mipmapping` may be any of the three values but must be `RT_FILTER_NONE` if the texture sampler contains only a single MIP level or one of `RT_FILTER_NEAREST` or `RT_FILTER_LINEAR` if the texture sampler contains more than one MIP level.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTextureSamplerSetFilteringModes` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetFilteringModes

1.12.31 `rtTextureSamplerSetIndexingMode`

NAME

`rtTextureSamplerSetIndexingMode` - Sets the indexing mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetIndexingMode(RTtexturesampler texturesampler,
                                         RTtextureindexmode indexmode)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`indexmode`

The new indexing mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetIndexingMode` sets the indexing mode of `texturesampler` to `indexmode`. `indexmode` can take on one of the following values:

```
RT_TEXTURE_INDEX_NORMALIZED_COORDINATES,
RT_TEXTURE_INDEX_ARRAY_INDEX
```

These values are used to control the interpretation of texture coordinates. If the index mode is set to `RT_TEXTURE_INDEX_NORMALIZED_COORDINATES`, the texture is parameterized over [0,1]. If the index mode is set to `RT_TEXTURE_INDEX_ARRAY_INDEX` then texture coordinates are interpreted as array indices into the contents of the underlying buffer objects.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_MEMORY_ALLOCATION_FAILED
```

HISTORY

`rtTextureSamplerSetIndexingMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetIndexingMode

1.12.32 `rtTextureSamplerSetMaxAnisotropy`

NAME

`rtTextureSamplerSetMaxAnisotropy` - Sets the maximum anisotropy of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetMaxAnisotropy(RTtexturesampler texturesampler,
                                          float value)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`value`

The new maximum anisotropy level of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetMaxAnisotropy` sets the maximum anisotropy of `texturesampler` to `value`. A float value greater than 0 will enable anisotropic filtering at the specified value.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerSetMaxAnisotropy` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetMaxAnisotropy

1.12.33 `rtTextureSamplerSetMipLevelCount`

NAME

`rtTextureSamplerSetMipLevelCount` - Sets the number of MIP levels in a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetMipLevelCount(RTtexturesampler texturesampler,
                                          unsigned int num_mip_levels)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`num_mip_levels`

The new number of MIP levels of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetMipLevelCount` sets the number of MIP levels in `texturesampler` to `num_mip_levels`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerSetMipLevelCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetMipLevelCount

1.12.34 rtTextureSamplerSetReadMode

NAME

rtTextureSamplerSetReadMode - Sets the read mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>

RTresult rtTextureSamplerSetReadMode(RTtexturesampler texturesampler,
                                     RTtexturereadmode readmode)
```

PARAMETERS

texturesampler

The texture sampler object to be changed.

readmode

The new read mode of the texture sampler.

DESCRIPTION

rtTextureSamplerSetReadMode sets the data read mode of **texturesampler** to **readmode**. **readmode** can take one of the following values:

```
RT_TEXTURE_READ_ELEMENT_TYPE
RT_TEXTURE_READ_NORMALIZED_FLOAT
```

readmode controls the returned value of the texture sampler when it is used to sample textures. **RT_TEXTURE_READ_ELEMENT_TYPE** will return data of the type of the underlying buffer objects. **RT_TEXTURE_READ_NORMALIZED_FLOAT** will return floating point values normalized by the range of the underlying type. If the underlying type is floating point, **RT_TEXTURE_READ_NORMALIZED_FLOAT** and **RT_TEXTURE_READ_ELEMENT_TYPE** are equivalent, always returning the unmodified floating point value.

For example, a texture sampler that samples a buffer of type **RT_FORMAT_UNSIGNED_BYTE** with a read mode of **RT_TEXTURE_READ_NORMALIZED_FLOAT** will convert integral values from the range [0,255] to floating point values in the range [0,1] automatically as the buffer is sampled from.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
```

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

RT_ERROR_MEMORY_ALLOCATION_FAILED

HISTORY

`rtTextureSamplerSetReadMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetReadMode

1.12.35 `rtTextureSamplerSetWrapMode`

NAME

`rtTextureSamplerSetWrapMode` - Sets the wrapping mode of a texture sampler.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtTextureSamplerSetWrapMode(RTtexturesampler texturesampler,  
                                     unsigned int dim,  
                                     RTwrapmode wrapmode)
```

PARAMETERS

`texturesampler`

The texture sampler object to be changed.

`wrapmode`

The new wrap mode of the texture sampler.

DESCRIPTION

`rtTextureSamplerSetWrapMode` sets the wrapping mode of `texturesampler` to `wrapmode` for the texture dimension specified by `dim`. `wrapmode` can take one of the following values:

```
RT_WRAP_REPEAT  
RT_WRAP_CLAMP_TO_EDGE
```

The wrapping mode controls the behavior of the texture sampler as texture coordinates wrap around the range specified by the indexing mode. These values mirror the OpenGL behavior of textures.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS  
RT_ERROR_INVALID_CONTEXT  
RT_ERROR_INVALID_VALUE  
RT_ERROR_MEMORY_ALLOCATION_FAILED
```


HISTORY

`rtTextureSamplerSetWrapMode` was introduced in *OptiX* 1.0.

SEE ALSO

rtTextureSamplerGetWrapMode

1.12.36 `rtTextureSamplerValidate`

NAME

`rtTextureSamplerValidate` - Validates the state of a texture sampler.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtTextureSamplerValidate(RTtexturesampler texturesampler)
```

PARAMETERS

`texturesampler`

The texture sampler to be validated.

DESCRIPTION

`rtTextureSamplerValidate` checks `texturesampler` for completeness. If `texturesampler` does not have buffers attached to all of its MIP levels and array slices or if the filtering modes are incompatible with the current MIP level and array slice configuration then the call will return `RT_ERROR_INVALID_CONTEXT`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtTextureSamplerValidate` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextValidate

1.13 Variables

NAME

Variable

DESCRIPTION

This section describes the API functions for creation and handling of Variable objects.

`rtVariableGet`

`rtVariableGetContext`

`rtVariableGetName`

`rtVariableGetAnnotation`

`rtVariableGetObject`

`rtVariableGetType`

`rtVariableGetUserData`

`rtVariableGetSize`

`rtVariableSet`

`rtVariableSetObject`

`rtVariableSetUserData`

HISTORY

Variable objects were introduced in *OptiX* 1.0.

SEE ALSO

Context, Geometry Group, Group Node, Selector Node, Transform Node, Acceleration Structure, Geometry Instance, Geometry, Material, Program, Buffer, Texture Sampler, Context-Free Functions

1.13.1 rtVariableGet

NAME

rtVariableGet - Returns the value of a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGet1f(RTvariable variable,
                        float* v1)

RTresult rtVariableGet2f(RTvariable variable,
                        float* v1,
                        float* v2)

RTresult rtVariableGet3f(RTvariable variable,
                        float* v1,
                        float* v2,
                        float* v3)

RTresult rtVariableGet4f(RTvariable variable,
                        float* v1,
                        float* v2,
                        float* v3,
                        float* v4)

RTresult rtVariableGet1i(RTvariable variable,
                        int* v1)

RTresult rtVariableGet2i(RTvariable variable,
                        int* v1,
                        int* v2)

RTresult rtVariableGet3i(RTvariable variable,
                        int* v1,
                        int* v2,
                        int* v3)

RTresult rtVariableGet4i(RTvariable variable,
                        int* v1,
                        int* v2,
                        int* v3,
                        int* v4)
```

```
RTresult rtVariableGet1ui(RTvariable variable,  
                          unsigned int* v1)
```

```
RTresult rtVariableGet2ui(RTvariable variable,  
                          unsigned int* v1,  
                          unsigned int* v2)
```

```
RTresult rtVariableGet3ui(RTvariable variable,  
                          unsigned int* v1,  
                          unsigned int* v2,  
                          unsigned int* v3)
```

```
RTresult rtVariableGet4ui(RTvariable variable,  
                          unsigned int* v1,  
                          unsigned int* v2,  
                          unsigned int* v3,  
                          unsigned int* v4)
```

PARAMETERS

variable

Specifies the program variable to be queried.

v1, v2, v3, v4

Returns the values of the program variable's components.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtVariableGet1fv(RTvariable variable,  
                          float* v)
```

```
RTresult rtVariableGet2fv(RTvariable variable,  
                          float* v)
```

```
RTresult rtVariableGet3fv(RTvariable variable,  
                          float* v)
```

```
RTresult rtVariableGet4fv(RTvariable variable,  
                          float* v)
```

```
RTresult rtVariableGet1iv(RTvariable variable,  
                          int* v)
```

```
RTresult rtVariableGet2iv(RTvariable variable,  
                          int* v)
```

```
RTresult rtVariableGet3iv(RTvariable variable,  
                          int* v)
```

```
RTresult rtVariableGet4iv(RTvariable variable,  
                          int* v)
```

```
RTresult rtVariableGet1uiv(RTvariable variable,  
                           unsigned int* v)
```

```
RTresult rtVariableGet2uiv(RTvariable variable,  
                           unsigned int* v)
```

```
RTresult rtVariableGet3uiv(RTvariable variable,  
                           unsigned int* v)
```

```
RTresult rtVariableGet4uiv(RTvariable variable,  
                           unsigned int* v)
```

PARAMETERS

variable

Specifies the program variable to be queried.

v

Returns the values of the program variable's components.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtVariableGetMatrix2x2fv(RTvariable variable,  
                                  int transpose,  
                                  float* m)
```

```
RTresult rtVariableGetMatrix2x3fv(RTvariable variable,  
                                  int transpose,  
                                  float* m)
```

```
RTresult rtVariableGetMatrix2x4fv(RTvariable variable,  
                                  int transpose,  
                                  float* m)
```

```
RTresult rtVariableGetMatrix3x2fv(RTvariable variable,  
                                  int transpose,  
                                  float* m)
```

```
RTresult rtVariableGetMatrix3x3fv(RTvariable variable,
                                  int transpose,
                                  float* m)
```

```
RTresult rtVariableGetMatrix3x4fv(RTvariable variable,
                                  int transpose,
                                  float* m)
```

```
RTresult rtVariableGetMatrix4x2fv(RTvariable variable,
                                  int transpose,
                                  float* m)
```

```
RTresult rtVariableGetMatrix4x3fv(RTvariable variable,
                                  int transpose,
                                  float* m)
```

```
RTresult rtVariableGetMatrix4x4fv(RTvariable variable,
                                  int transpose,
                                  float* m)
```

PARAMETERS

variable

Specifies the program variable to be queried.

transpose

Specifies whether to transpose the matrix as the values are returned from the program variable.

m

Returns the values of the program variable's matrix.

DESCRIPTION

rtVariableGet returns the value of a program variable or variable array. The target variable is specified by **variable**.

The commands **rtVariableGet{1-2-3-4}{f-i-ui}v** are used to query the value of a program variable specified by **variable** using the pointers passed as return locations for each component of the vector-typed variable. The number specified in the command should match the number of components in the data type of the specified program variable (e.g., 1 for float, int, unsigned int; 2 for float2, int2, uint2, etc.). The suffix **f** indicates that floating-point values are expected to be returned, the suffix **i** indicates that integer values are expected, and the suffix **ui** indicates that unsigned integer values are expected, and this type should also match the data type of the specified program variable. The **f** variants of this function should be used to query values for program variables defined as float, float2, float3, float4, or arrays of these. The **i** variants of this function should be used to query values for program variables defined as int, int2, int3, int4, or arrays of these. The **ui** variants of this function should be used to query values for program variables defined as unsigned int, uint2, uint3, uint4, or arrays of these. The **v** variants of this function should be

used to return the program variable's value to the array specified by parameter **v**. In this case, the array **v** should be large enough to accommodate all of the program variable's components.

The commands **rtVariableGetMatrix{2-3-4}x{2-3-4}fv** are used to query the value of a program variable whose data type is a matrix. The numbers in the command names are interpreted as the dimensionality of the matrix. For example, **2x4** indicates a 2 x 4 matrix with 2 columns and 4 rows (i.e., 8 values). If **transpose** is **0**, the matrix is returned in row major order, otherwise in column major order.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

rtVariableGet was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableSet, *rtVariableGetType*, *rtContextDeclareVariable*

1.13.2 `rtVariableGetContext`

NAME

`rtVariableGetContext` - Returns the context associated with a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetContext(RTvariable variable,
                             RTcontext* context)
```

PARAMETERS

`variable`

Specifies the program variable to be queried.

`context`

Returns the context associated with the program variable.

DESCRIPTION

`rtVariableGetContext` queries the context associated with a program variable. The target variable is specified by `variable`. The context of the program variable is returned to `*context` if the pointer `context` is not NULL. If `variable` is not a valid variable, `*context` is set to NULL and `RT_ERROR_INVALID_VALUE` is returned.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtVariableGetContext` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextDeclareVariable

1.13.3 `rtVariableGetName`

NAME

`rtVariableGetName` - Queries the name of a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetName(RTvariable variable,
                          const char** name_return)
```

PARAMETERS

`variable`

Specifies the program variable to be queried.

`name_return`

Returns the program variable's name.

DESCRIPTION

`rtVariableGetName` queries a program variable's name. The variable of interest is specified by `variable`, which should be a value returned by `rtContextDeclareVariable`. A pointer to the string containing the name of the variable shall be returned to the location pointed to by the pointer `name_return`. If `variable` is not a valid variable, this call sets `*name_return` to `NULL` and returns `RT_ERROR_INVALID_VALUE`. `*name_return` will point to valid memory until another API function that returns a string is called.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

`rtVariableGetName` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextDeclareVariable

1.13.4 rtVariableGetAnnotation

NAME

rtVariableGetAnnotation - Queries the annotation string of a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetAnnotation(RTvariable variable,
                                const char** annotation_return)
```

PARAMETERS

variable

Specifies the program variable to be queried.

annotation_return

Returns the program variable's annotation string.

DESCRIPTION

rtVariableGetAnnotation queries a program variable's annotation string. A pointer to the string containing the annotation shall be returned to the location pointed to by the pointer **annotation_return**. If **variable** is not a valid variable, this call sets ***annotation_return** to NULL and returns `RT_ERROR_INVALID_VALUE`. ***annotation_return** will point to valid memory until another API function that returns a string is called.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

HISTORY

rtVariableGetAnnotation was introduced in *OptiX* 1.0.

SEE ALSO

rtContextDeclareVariable, *rtContextDeclareAnnotation*

1.13.5 `rtVariableGetObject`

NAME

`rtVariableGetObject` - Returns the value of a *OptiX* object program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetObject(RTvariable variable,
                             RTOBJECT* object)
```

PARAMETERS

`variable`

Specifies the program variable to be queried.

`object`

Returns the value of the program variable.

DESCRIPTION

`rtVariableGetObject` queries the value of a program variable whose data type is a *OptiX* object. The target variable is specified by `variable`. The value of the program variable is returned in the location pointed to by `object`. The concrete type of the program variable can be queried using `rtVariableGetType`, and the `RTOBJECT` handle returned by `rtVariableGetObject` may safely be cast to an *OptiX* handle of corresponding type. If `variable` is not a valid variable, this call sets the location pointed to by `object` to `NULL` and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtVariableGetObject` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableSetObject, rtVariableGetType, rtContextDeclareVariable

1.13.6 `rtVariableGetSize`

NAME

`rtVariableGetSize` - Queries the size, in bytes, of a variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetSize(RTvariable variable,
                           RTsize* size)
```

PARAMETERS

`variable`

Specifies the program variable to be queried.

`size`

Specifies a pointer where the size of the variable, in bytes, will be returned.

DESCRIPTION

`rtVariableGetSize` queries a declared program variable for its size in bytes. This is most often used to query the size of a variable that has a user-defined type. Builtin types (int, float, unsigned int, etc.) may be queried, but object typed variables, such as buffers, texture samplers and graph nodes, cannot be queried and will return `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtVariableGetSize` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableGetUserData, *rtContextDeclareVariable*

1.13.7 `rtVariableGetType`

NAME

`rtVariableGetType` - Returns type information about a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetType(RTvariable variable,
                          RObjecttype* type_return)
```

PARAMETERS

`variable`

Specifies the program variable to be queried.

`type_return`

Returns the type of the program variable.

DESCRIPTION

`rtVariableGetType` queries a program variable's type. The variable of interest is specified by `variable`. The enumeration identifying the type of the program variable shall be returned to the location pointed to by `type_return`, if it is not equal to NULL. In this case, after `rtVariableGetType`, the location pointed to by `type_return` shall be one of the following:

```
RT_OBJECTTYPE_UNKNOWN
RT_OBJECTTYPE_GROUP
RT_OBJECTTYPE_GEOMETRY_GROUP
RT_OBJECTTYPE_TRANSFORM
RT_OBJECTTYPE_SELECTOR
RT_OBJECTTYPE_GEOMETRY_INSTANCE
RT_OBJECTTYPE_BUFFER
RT_OBJECTTYPE_TEXTURE_SAMPLER
RT_OBJECTTYPE_OBJECT
RT_OBJECTTYPE_MATRIX_FLOAT2x2
RT_OBJECTTYPE_MATRIX_FLOAT2x3
RT_OBJECTTYPE_MATRIX_FLOAT2x4
RT_OBJECTTYPE_MATRIX_FLOAT3x2
RT_OBJECTTYPE_MATRIX_FLOAT3x3
RT_OBJECTTYPE_MATRIX_FLOAT3x4
RT_OBJECTTYPE_MATRIX_FLOAT4x2
RT_OBJECTTYPE_MATRIX_FLOAT4x3
RT_OBJECTTYPE_MATRIX_FLOAT4x4
RT_OBJECTTYPE_FLOAT
```

```
RT_OBJECTTYPE_FLOAT2
RT_OBJECTTYPE_FLOAT3
RT_OBJECTTYPE_FLOAT4
RT_OBJECTTYPE_INT
RT_OBJECTTYPE_INT2
RT_OBJECTTYPE_INT3
RT_OBJECTTYPE_INT4
RT_OBJECTTYPE_UNSIGNED_INT
RT_OBJECTTYPE_UNSIGNED_INT2
RT_OBJECTTYPE_UNSIGNED_INT3
RT_OBJECTTYPE_UNSIGNED_INT4
RT_OBJECTTYPE_USER
```

If **variable** is not valid, this call returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
```

HISTORY

`rtVariableGetType` was introduced in *OptiX* 1.0.

SEE ALSO

rtContextDeclareVariable

1.13.8 `rtVariableGetUserData`

NAME

`rtVariableGetUserData` - Returns the value of a program variable whose data type is user-defined.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableGetUserData(RTvariable variable,
                               RTsize size,
                               void* ptr)
```

PARAMETERS

variable

Specifies the program variable to be queried.

size

Specifies the size of the program variable, in bytes.

ptr

The target memory location where to copy the value of the variable.

DESCRIPTION

`rtVariableGetUserData` queries the value of a program variable whose data type is user-defined. The variable of interest is specified by **variable**. The size of the variable's value must match the value given by the parameter **size**. The value of the program variable is copied to the memory region pointed to by **ptr**. The storage at location **ptr** must be large enough to accommodate all of the program variable's value data. If **variable** is not a valid variable, this call has no effect and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtVariableGetUserData` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableSetUserData, rtContextDeclareVariable

1.13.9 rtVariableSet

NAME

rtVariableSet - Modifies the value of a program variable.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableSet1f(RTvariable variable,
                        float v1)

RTresult rtVariableSet2f(RTvariable variable,
                        float v1,
                        float v2)

RTresult rtVariableSet3f(RTvariable variable,
                        float v1,
                        float v2,
                        float v3)

RTresult rtVariableSet4f(RTvariable variable,
                        float v1,
                        float v2,
                        float v3,
                        float v4)

RTresult rtVariableSet1i(RTvariable variable,
                        int v1)

RTresult rtVariableSet2i(RTvariable variable,
                        int v1,
                        int v2)

RTresult rtVariableSet3i(RTvariable variable,
                        int v1,
                        int v2,
                        int v3)

RTresult rtVariableSet4i(RTvariable variable,
                        int v1,
                        int v2,
                        int v3,
                        int v4)
```

```
RTresult rtVariableSet1ui(RTvariable variable,  
                          unsigned int v1)
```

```
RTresult rtVariableSet2ui(RTvariable variable,  
                          unsigned int v1,  
                          unsigned int v2)
```

```
RTresult rtVariableSet3ui(RTvariable variable,  
                          unsigned int v1,  
                          unsigned int v2,  
                          unsigned int v3)
```

```
RTresult rtVariableSet4ui(RTvariable variable,  
                          unsigned int v1,  
                          unsigned int v2,  
                          unsigned int v3,  
                          unsigned int v4)
```

PARAMETERS

variable

Specifies the program variable to be modified.

v1, v2, v3, v4

Specify the new values of the program variable's components.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtVariableSet1fv(RTvariable variable,  
                         const float* v)
```

```
RTresult rtVariableSet2fv(RTvariable variable,  
                         const float* v)
```

```
RTresult rtVariableSet3fv(RTvariable variable,  
                         const float* v)
```

```
RTresult rtVariableSet4fv(RTvariable variable  
                         const float* v)
```

```
RTresult rtVariableSet1iv(RTvariable variable  
                         const int* v)
```

```
RTresult rtVariableSet2iv(RTvariable variable  
                         const int* v)
```

```
RTresult rtVariableSet3iv(RTvariable variable
                          const int* v)

RTresult rtVariableSet4iv(RTvariable variable
                          const int* v)

RTresult rtVariableSet1uiv(RTvariable variable
                           const unsigned int* v)

RTresult rtVariableSet2uiv(RTvariable variable,
                           const unsigned int* v)

RTresult rtVariableSet3uiv(RTvariable variable,
                           const unsigned int* v)

RTresult rtVariableSet4uiv(RTvariable variable,
                           const unsigned int* v)
```

PARAMETERS

variable

Specifies the program variable to be modified.

v

Specifies a pointer to the new values of the program variable's components.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableSetMatrix2x2fv(RTvariable variable,
                                  int transpose,
                                  const float* m)

RTresult rtVariableSetMatrix2x3fv(RTvariable variable,
                                  int transpose,
                                  const float* m)

RTresult rtVariableSetMatrix2x4fv(RTvariable variable,
                                  int transpose,
                                  const float* m)

RTresult rtVariableSetMatrix3x2fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

```
RTresult rtVariableSetMatrix3x3fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

```
RTresult rtVariableSetMatrix3x4fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

```
RTresult rtVariableSetMatrix4x2fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

```
RTresult rtVariableSetMatrix4x3fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

```
RTresult rtVariableSetMatrix4x4fv(RTvariable variable,
                                  int transpose,
                                  const float* m)
```

DESCRIPTION

rtVariableSet modifies the value of a program variable or variable array. The target variable is specified by **variable**, which should be a value returned by **rtContextGetVariable**.

The commands **rtVariableSet{1-2-3-4}{f-i-ui}v** are used to modify the value of a program variable specified by **variable** using the values passed as arguments. The number specified in the command should match the number of components in the data type of the specified program variable (e.g., 1 for float, int, unsigned int; 2 for float2, int2, uint2, etc.). The suffix **f** indicates that **variable** has floating point type, the suffix **i** indicates that **variable** has integral type, and the suffix **ui** indicates that that **variable** has unsigned integral type. The **v** variants of this function should be used to load the program variable's value from the array specified by parameter **v**. In this case, the array **v** should contain as many elements as there are program variable components.

The commands **rtVariableSetMatrix{2-3-4}x{2-3-4}fv** are used to modify the value of a program variable whose data type is a matrix. The numbers in the command names are interpreted as the dimensionality of the matrix. For example, **2x4** indicates a 2 x 4 matrix with 2 columns and 4 rows (i.e., 8 values). If **transpose** is **0**, the matrix is specified in row major order, otherwise in column major order.

If **variable** is not a valid variable, these calls have no effect and return **RT_ERROR_INVALID_VALUE**.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_CONTEXT

RT_ERROR_INVALID_VALUE

HISTORY

`rtVariableSet` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableGet, *rtVariableGetType*, *rtContextDeclareVariable*

1.13.10 `rtVariableSetObject`

NAME

`rtVariableSetObject` - Sets a program variable value to a *OptiX* object.

SYNOPSIS

```
#include <optix.h>

RTresult rtVariableSetObject(RTvariable variable,
                             RTOBJECT object)
```

PARAMETERS

`variable`

Specifies the program variable to be set.

`object`

Specifies the new value of the program variable.

DESCRIPTION

`rtVariableSetObject` sets a program variable to an *OptiX* object value. The target variable is specified by `variable`. The new value of the program variable is specified by `object`. The concrete type of `object` can be one of `RTbuffer`, `RTtexturesampler`, `RTgroup`, `RTselector`, `RTgeometrygroup`, or `RTtransform`. If `variable` is not a valid variable or `object` is not a valid *OptiX* object, this call has no effect and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

```
RT_SUCCESS
RT_ERROR_INVALID_CONTEXT
RT_ERROR_INVALID_VALUE
RT_ERROR_TYPE_MISMATCH
```

HISTORY

`rtVariableSetObject` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableGetObject, rtContextDeclareVariable

1.13.11 `rtVariableSetUserData`

NAME

`rtVariableSetUserData` - Modifies the value of a program variable whose data type is user-defined.

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtVariableSetUserData(RTvariable variable,  
                               RTsize size,  
                               const void* ptr)
```

PARAMETERS

variable

Specifies the program variable to be modified.

size

Specifies the size of the new value, in bytes.

ptr

Specifies a pointer to the new value of the program variable.

DESCRIPTION

`rtVariableSetUserData` modifies the value of a program variable whose data type is user-defined. The value copied into the variable is defined by an arbitrary region of memory, pointed to by **ptr**. The size of the memory region is given by **size**. The target variable is specified by **variable**. If **variable** is not a valid variable, this call has no effect and returns `RT_ERROR_INVALID_VALUE`.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_CONTEXT`

`RT_ERROR_INVALID_VALUE`

`RT_ERROR_MEMORY_ALLOCATION_FAILED`

`RT_ERROR_TYPE_MISMATCH`

HISTORY

`rtVariableSetUserData` was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableGetUserData, *rtContextDeclareVariable*

1.14 Context Free Functions

NAME

Context-Free Functions

DESCRIPTION

This section describes general *OptiX* API functions that are not related to a specific context.

rtDeviceGetAttribute

rtDeviceGetDeviceCount

rtDeviceGetD3D9Device

rtDeviceGetD3D10Device

rtDeviceGetD3D11Device

rtDeviceGetWGLDevice

rtGetVersion

HISTORY

Context-free functions were introduced in *OptiX* 1.0.

SEE ALSO

Context, *Geometry Group*, *Group Node*, *Selector Node*, *Transform Node*, *Acceleration Structure*, *Geometry Instance*, *Geometry*, *Material*, *Program*, *Buffer*, *Texture Sampler*, *Variables*

1.14.1 rtDeviceGetAttribute

NAME

rtDeviceGetAttribute - returns an attribute specific to an *OptiX* device.

SYNOPSIS

```
#include <optix.h>

RTresult RTAPI rtDeviceGetAttribute(int ordinal,
                                   RTdeviceattribute attrib,
                                   RTsize size,
                                   void* p);
```

PARAMETERS

ordinal

OptiX device ordinal.

attrib

Attribute to query.

size

Size of the attribute being queried. Parameter **p** must have at least this much memory backing it.

p

Return pointer where the value of the attribute will be copied into. This must point to at least **size** bytes of memory.

DESCRIPTION

rtDeviceGetAttribute() returns in **p** the value of the per device attribute specified by **attrib** for device **ordinal**.

Each attribute can have a different size. The sizes are given in the following list:

RT_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK	sizeof(int)
RT_DEVICE_ATTRIBUTE_CLOCK_RATE	sizeof(int)
RT_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT	sizeof(int)
RT_DEVICE_ATTRIBUTE_EXECUTION_TIMEOUT_ENABLED	sizeof(int)
RT_DEVICE_ATTRIBUTE_MAX_HARDWARE_TEXTURE_COUNT	sizeof(int)
RT_DEVICE_ATTRIBUTE_NAME	upto B<size>-1
RT_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY	sizeof(int2)
RT_DEVICE_ATTRIBUTE_TOTAL_MEMORY	sizeof(RTsize)

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE - Can be returned if **size** does not match the proper size of the attribute, if **p** is NULL, or if **ordinal** does not correspond to an *OptiX* device.

HISTORY

`rtDeviceGetAttribute` was introduced in *OptiX* 2.0.

SEE ALSO

rtDeviceGetDeviceCount, *rtContextGetAttribute*

1.14.2 `rtDeviceGetDeviceCount`

NAME

`rtDeviceGetDeviceCount` - returns the number of *OptiX* capable devices

SYNOPSIS

```
#include <optix.h>
```

```
RTresult rtDeviceGetDeviceCount(unsigned int* count)
```

PARAMETERS

`count`

Number devices available for *OptiX*.

DESCRIPTION

`rtDeviceGetDeviceCount()` returns in `count` the number of compute devices that are available in the host system and will be used by *OptiX*.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtDeviceGetDeviceCount` was introduced in *OptiX* 1.0.

SEE ALSO

rtGetVersion

1.14.3 rtDeviceGetD3D9Device

NAME

rtDeviceGetD3D9Device - returns the *OptiX* device number associated with the specified name of a D3D9 adapter.

SYNOPSIS

```
#include <optix_d3d9_interop.h>

RTresult rtDeviceGetD3D9Device(int* device,
                               const char* pszAdapterName)
```

PARAMETERS

device

A handle to the memory location where the *OptiX* device ordinal associated with **pszAdapterName** will be stored.

pszAdapterName

The name of an adapter as can be found in the DeviceName field in the D3DADAPTER_IDENTIFIER9 struct.

DESCRIPTION

rtDeviceGetD3D9Device() returns in **device** the *OptiX* device ID of the adapter represented by **pszAdapterName**. **pszAdapterName** is the DeviceName field in the D3DADAPTER_IDENTIFIER9 struct. In combination with **rtContextSetDevices()**, this function can be used to restrict *OptiX* to use only one device. The same device the D3D9 commands will be sent to.

This function is only supported on Windows platforms.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtDeviceGetD3D9Device was introduced in *OptiX* 2.5.

SEE ALSO

rtDeviceGetDeviceCount

1.14.4 rtDeviceGetD3D10Device

NAME

rtDeviceGetD3D10Device - returns the *OptiX* device number associated with the pointer to a D3D10 adapter.

SYNOPSIS

```
#include <optix_d3d10_interop.h>

RtResult rtDeviceGetD3D10Device(int* device,
                                ID3D10Device* d3d10Device)
```

PARAMETERS

device

A handle to the memory location where the *OptiX* device ordinal associated with **d3d10Device** will be stored.

d3d10Device

A pointer to an ID3D10Device as returned from D3D10CreateDeviceAndSwapChain.

DESCRIPTION

rtDeviceGetD3D10Device() returns in **device** the *OptiX* device ID of the adapter represented by **d3d10Device**. **d3d10Device** is a pointer returned from D3D10CreateDeviceAndSwapChain. In combination with **rtContextSetDevices()**, this function can be used to restrict *OptiX* to use only one device. The same device the D3D10 commands will be sent to.

This function is only supported on Windows platforms.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtDeviceGetD3D10Device was introduced in *OptiX* 2.5.

SEE ALSO

rtDeviceGetDeviceCount

1.14.5 rtDeviceGetD3D11Device

NAME

rtDeviceGetD3D11Device - returns the *OptiX* device number associated with the pointer to a D3D11 adapter.

SYNOPSIS

```
#include <optix_d3d11_interop.h>

RtResult rtDeviceGetD3D11Device(int* device,
                                ID3D11Device* d3d11Device)
```

PARAMETERS

device

A handle to the memory location where the *OptiX* device ordinal associated with **d3d11Device** will be stored.

d3d11Device

A pointer to an ID3D11Device as returned from D3D11CreateDeviceAndSwapChain.

DESCRIPTION

rtDeviceGetD3D11Device() returns in **device** the *OptiX* device ID of the adapter represented by **d3d11Device**. **d3d11Device** is a pointer returned from D3D11CreateDeviceAndSwapChain. In combination with **rtContextSetDevices()**, this function can be used to restrict *OptiX* to use only one device. The same device the D3D11 commands will be sent to.

This function is only supported on Windows platforms.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtDeviceGetD3D11Device was introduced in *OptiX* 2.5.

SEE ALSO

rtDeviceGetDeviceCount

1.14.6 rtDeviceGetWGLDevice

NAME

rtDeviceGetWGLDevice - returns the *OptiX* device number associated with the specified GPU.

SYNOPSIS

```
#include <optix_gl_interop.h>

RTresult rtDeviceGetWGLDevice(int* device,
                              HGPUNV hGpu)
```

PARAMETERS

device

A handle to the memory location where the *OptiX* device ordinal associated with **hGpu** will be stored.

hGpu

A handle to a GPU as returned from the WGL_NV_gpu_affinity OpenGL extension.

DESCRIPTION

rtDeviceGetWGLDevice() returns in **device** the *OptiX* device ID of the GPU represented by **hGpu**. **hGpu** is returned from WGL_NV_gpu_affinity, an OpenGL extension. This enables **OptiX** to create a context on the same GPU that OpenGL commands will be sent to, improving OpenGL interoperation efficiency.

This function is only supported on Windows platforms.

RETURN VALUES

Relevant return values:

RT_SUCCESS

RT_ERROR_INVALID_VALUE

HISTORY

rtDeviceGetWGLDevice was introduced in *OptiX* 1.0.

SEE ALSO

rtDeviceGetDeviceCount, WGL_NV_gpu_affinity

1.14.7 `rtGetVersion`

NAME

`rtGetVersion` - returns the current *OptiX* version

SYNOPSIS

```
#include <optix.h>

RTresult rtGetVersion(unsigned int* version)
```

PARAMETERS

`version`

OptiX version number.

DESCRIPTION

`rtGetVersion()` returns in `version` a numerically comparable version number of the current *OptiX* library.

RETURN VALUES

Relevant return values:

`RT_SUCCESS`

`RT_ERROR_INVALID_VALUE`

HISTORY

`rtGetVersion` was introduced in *OptiX* 1.0.

SEE ALSO

rtDeviceGetDeviceCount

Chapter 2

CUDA C Reference

2.1 Declarations

2.1.1 `rtDeclareAnnotation`

NAME

`rtDeclareAnnotation` - Annotation declaration.

SYNOPSIS

```
#include <optix.h>

rtDeclareAnnotation(name,
                    annotation)
```

DESCRIPTION

`rtDeclareAnnotation` sets the annotation **annotation** of the given variable **name**. Typically annotations are declared using an argument to `rtDeclareVariable`, but variables of type `rtBuffer` and `rtTextureSampler` are declared using templates, so separate annotation attachment is required.

OptiX does not attempt to interpret the annotation in any way. It is considered metadata for the application to query and interpret in its own way.

Valid annotations

The macro `rtDeclareAnnotation` uses the C pre-processor's "stringification" feature to turn the literal text of the annotation argument into a string constant. The pre-processor will backslash-escape quotes and backslashes within the text of the annotation. Leading and trailing whitespace will be ignored, and sequences of whitespace in the middle of the text is converted to a single space character in the result. The only restriction the C-PP places on the text is that it may not contain a comma character unless it is either quoted or contained within parens: `"`, `"` or `(`, `)`.

Example(s):

```
rtDeclareAnnotation( tex, this is a test );
annotation = "this is a test"

rtDeclareAnnotation( tex, "this is a test" );
annotation = "\"this is a test\""

rtDeclareAnnotation( tex, float3 a = {1, 2, 3} );
--> Compile Error, no unquoted commas may be present in the annotation

rtDeclareAnnotation( tex, "float3 a = {1, 2, 3}" );
annotation = "\"float3 a = {1, 2, 3}\""

rtDeclareAnnotation( tex, string UIWidget = "slider";
```

```
        float UIMin = 0.0;
        float UIMax = 1.0; );
annotation = "string UIWidget = \"slider\"; float UIMin = 0.0; float UIMax = 1.0;"
```

HISTORY

rtDeclareAnnotation was introduced in *OptiX* 1.0.

SEE ALSO

rtDeclareVariable, *rtVariableGetAnnotation*

2.1.2 `rtDeclareVariable`

NAME

`rtDeclareVariable` - Variable declaration.

SYNOPSIS

```
#include <optix.h>

rtDeclareVariable(type,
                  name,
                  semanticName,
                  annotation)
```

DESCRIPTION

`rtDeclareVariable` declares variable **name** of the specified **type**. By default, the variable name will be matched against a variable declared on the API object using the lookup hierarchy for the current program. Using the `semanticName`, this variable can be bound to internal state, to the payload associated with a ray, or to attributes that are communicated between intersection and material programs. An additional optional annotation can be used to associate application-specific metadata with the variable as well.

Type may be a primitive type or a user-defined struct (See `rtVariableSetUserData`). Except for the ray payload and attributes, the declared variable will be read-only. The variable will be visible to all of the cuda functions defined in the current file. The binding of variables to values on API objects is allowed to vary from one instance to another.

Valid semanticNames

`rtLaunchIndex`

`rtLaunchIndex` is the launch invocation index. Type must be one of `unsigned int`, `uint2`, `uint3`, `int`, `int2`, `int3` and is read-only.

`rtLaunchDim`

`rtLaunchDim` is the size of each dimension of the launch. The values range from 1 to the launch size in that dimension. Type must be one of `unsigned int`, `uint2`, `uint3`, `int`, `int2`, `int3` and is read-only.

`rtCurrentRay`

`rtCurrentRay` is the currently active ray, valid only when a call to `rtTrace` is active. Type must be `optix::Ray` and is read-only.

`rtIntersectionDistance`

`rtIntersectionDistance` The current closest hit distance, valid only when a call to `rtTrace` is active. Type must be `float` and is read-only.

rtRayPayload

rtRayPayload refers to the struct passed into the most recent **rtTrace** call and is read-write.

attribute name

attribute name refers to a named attribute passed from the intersect program to a closest-hit or any-hit program. The types must match in both sets of programs. This variable is read-only in the closest-hit or any-hit program and is written in the intersection program.

Valid annotations

The macro *rtDeclareVariable* uses the C pre-processor's "stringification" feature to turn the literal text of the annotation argument into a string constant. The pre-processor will backslash-escape quotes and backslashes within the text of the annotation. Leading and trailing whitespace will be ignored, and sequences of whitespace in the middle of the text is converted to a single space character in the result. The only restriction the C-PP places on the text is that it may not contain a comma character unless it is either quoted or contained within parens: "," or (,).

Example(s):

```
rtDeclareVariable( float3, var1, , this is a test );
annotation = "this is a test"
```

```
rtDeclareVariable( float3, var1, , "this is a test" );
annotation = "\"this is a test\""
```

```
rtDeclareVariable( float3, var1, , float3 a = {1, 2, 3} );
--> Compile Error, no unquoted commas may be present in the annotation
```

```
rtDeclareVariable( float3, var1, , "float3 a = {1, 2, 3}" );
annotation = "\"float3 a = {1, 2, 3}\""
```

```
rtDeclareAnnotation( tex, string UIWidget = "slider";
                    float UIMin = 0.0;
                    float UIMax = 1.0; );
annotation = "string UIWidget = \"slider\"; float UIMin = 0.0; float UIMax = 1.0;"
```

HISTORY

rtDeclareVariable was introduced in *OptiX* 1.0.

rtLaunchDim was introduced in *OptiX* 2.0.

SEE ALSO

rtDeclareAnnotation, **rtVariableGetAnnotation**, **rtContextDeclareVariable**, **rtProgramDeclareVariable**, **rtGroupDeclareVariable**, **rtSelectorDeclareVariable**, **rtGeometryInstanceDeclareVariable**, **rtGeometryDeclareVariable**, **rtMaterialDeclareVariable**

2.1.3 RT FUNCTION

NAME

RT_FUNCTION - Define an inline function to be called from within an OptiX program.

SYNOPSIS

```
#include <optix.h>
```

```
RT_FUNCTION return_value function_name(arguments) {...}
```

DESCRIPTION

RT_FUNCTION defines the function **function_name** with the specified return value and arguments that can be called from within a CUDA C-based program using typical C calling semantics.

HISTORY

RT_FUNCTION was introduced in *OptiX* 1.0.

SEE ALSO

RT_PROGRAM

2.1.4 RT PROGRAM

NAME

RT_PROGRAM - Define an OptiX program.

SYNOPSIS

```
#include <optix.h>
```

```
RT_PROGRAM void program_name(arguments) {...}
```

DESCRIPTION

RT_PROGRAM defines a program **program_name** with the specified arguments and return value. This function can be bound to a specific program object using **rtProgramCreateFromPTXString** or **rtProgramCreateFromFile**, which will subsequently get bound to different programmable binding points.

All programs should have a "void" return type. Bounding box programs will have an argument for the primitive index and the bounding box reference return value (type **nvrt::AAbb&**). Intersection programs will have a single int primitiveIndex argument. All other programs take zero arguments.

HISTORY

RT_PROGRAM was introduced in *OptiX* 1.0.

SEE ALSO

RT_FUNCTION, *rtProgramCreateFromPTXFile*, *rtProgramCreateFromPTXString*

2.2 Types

2.2.1 Aabb

NAME

Aabb - Axis-aligned bounding box

SYNOPSIS

```
#include <optixu_aabb.h>

class Aabb;
```

DESCRIPTION

Aabb is a utility class for computing and manipulating axis-aligned bounding boxes (aabbs). **Aabb** is primarily useful in the bounding box program associated with geometry objects. **Aabb** may also be useful in other computation and can be used in both host and device code.

METHODS

```
// Construct an invalid box.
Aabb();

// Construct from min and max vectors.
Aabb( const float3& min, const float3& max );

// Construct from three points (e.g. triangle).
Aabb( const float3& v0, const float3& v1, const float3& v2 );

// Array access.
float3& operator[]( int i );

// Const array access.
const float3& operator[]( int i ) const;

// Set using two vectors.
void set( const float3& min, const float3& max );

// Set using three points (e.g. triangle).
void set( const float3& v0, const float3& v1, const float3& v2 );

// Invalidate the box.
void invalidate();
```

```
// Check if the box is valid.
bool valid() const;

// Extend the box to include the given point.
void include( const float3& p );

// Extend the box to include the given box.
void include( const Aabb& other );

// Extend the box to include the given point.
void include( const float3& min, const float3& max );

// Compute the box center.
float3 center() const;

// Compute the box center in the given dimension.
float center( int dim ) const;

// Compute the box extent.
float3 extent() const;

// Compute the box extent in the given dimension.
float extent( int dim ) const;

// Compute the surface area of the box.
float area() const;

// Compute half the surface area of the box.
float halfArea() const;

// Get the index of the longest axis.
int longestAxis() const;

// Check for intersection with another box.
bool intersects( const Aabb& other ) const;

// Make the current box be the intersection between this one and another one.
void intersection( const Aabb& other );
```

MEMBERS

```
// Min and max bounds.
float3 m_min;
float3 m_max;
```

HISTORY

Aabb was introduced in *OptiX* 1.0.

SEE ALSO

RT_PROGRAM, *rtSetBoundingBoxProgram*

2.2.2 Matrix

NAME

Matrix - Matrix utility class.

SYNOPSIS

```
#include <optixu_matrix.h>

template <unsigned int M, unsigned int N> class Matrix;
typedef Matrix<2, 2> Matrix2x2;
typedef Matrix<2, 3> Matrix2x3;
typedef Matrix<2, 4> Matrix2x4;
typedef Matrix<3, 2> Matrix3x2;
typedef Matrix<3, 3> Matrix3x3;
typedef Matrix<3, 4> Matrix3x4;
typedef Matrix<4, 2> Matrix4x2;
typedef Matrix<4, 3> Matrix4x3;
typedef Matrix<4, 4> Matrix4x4;
```

DESCRIPTION

Matrix provides a utility class for small-dimension floating-point matrices, such as transformation matrices. **Matrix** may also be useful in other computation and can be used in both host and device code. Typedefs are provided for 2x2 through 4x4 matrices.

TYPES

```
typedef typename VectorDim<N>::VectorType floatN; // A row of the matrix
typedef typename VectorDim<M>::VectorType floatM; // A column of the matrix
```

METHODS and FUNCTIONS

```
// Create an uninitialized matrix.
Matrix();

// Create a matrix from the specified float array.
explicit Matrix( const float data[M*N] );

// Copy the matrix.
Matrix( const Matrix& m );

// Assignment operator.
Matrix& operator=( const Matrix& b );
```

```

// Access the specified element 0..N*M-1
float operator[]( unsigned int i )const;
float& operator[]( unsigned int i );

// Access the specified row 0..M. Returns float, float2, float3 or float4
// depending on the matrix size.
floatN getRow( unsigned int m )const;

// Access the specified column 0..N. Returns float, float2, float3 or float4
// depending on the matrix size.
floatM getCol( unsigned int n )const;

// Returns a pointer to the internal data array.
// The data array is stored in row-major order.
float* getData();

// Returns a const pointer to the internal data array.
// The data array is stored in row-major order.
const float* getData() const;

// Returns the transpose of the matrix.
Matrix<N,M> transpose();

// Returns the identity matrix.
static Matrix<N,N> identity();

// Ordered comparison operator so that the matrix can be used in an STL container.
bool operator<( const Matrix<M, N>& rhs ) const;

// Subtract two matrices of the same size.
template<unsigned int M, unsigned int N>
Matrix<M,N> operator-(const Matrix<M,N>& m1, const Matrix<M,N>& m2);

// Subtract two matrices of the same size.
template<unsigned int M, unsigned int N>
Matrix<M,N>& operator--(Matrix<M,N>& m1, const Matrix<M,N>& m2);

// Add two matrices of the same size.
template<unsigned int M, unsigned int N>
Matrix<M,N> operator+(const Matrix<M,N>& m1, const Matrix<M,N>& m2);

// Add two matrices of the same size.
template<unsigned int M, unsigned int N>
Matrix<M,N>& operator+=(Matrix<M,N>& m1, const Matrix<M,N>& m2);

// Multiply two compatible matrices.
template<unsigned int M, unsigned int N, unsigned int R>
Matrix<M,R> operator*( const Matrix<M,N>& m1, const Matrix<N,R>& m2)

```

```

// Multiply two compatible matrices.
template<unsigned int M>
Matrix<M,N>& operator*=(Matrix<M,M>& m1, const Matrix<M,M>& m2)

// Multiply two compatible matrices.
template<unsigned int M, unsigned int N>
typename Matrix<M,N>::floatM operator*(const Matrix<M,N>& m,
                                       const typename Matrix<M,N>::floatN& vec);

// Multiply two compatible matrices.
template<unsigned int M, unsigned int N>
typename Matrix<M,N>::floatN operator*(const typename Matrix<M,N>::floatM& vec,
                                       const Matrix<M,N>& m);

// Multiply matrix by a scalar.
template<unsigned int M, unsigned int N>
Matrix<M,N> operator*(const Matrix<M,N>& m, float f);

// Multiply matrix by a scalar.
template<unsigned int M, unsigned int N>
Matrix<M,N>& operator*=(Matrix<M,N>& m, float f);

// Multiply matrix by a scalar.
template<unsigned int M, unsigned int N>
Matrix<M,N> operator*(float f, const Matrix<M,N>& m);

// Divide matrix by a scalar.
template<unsigned int M, unsigned int N>
Matrix<M,N> operator/(const Matrix<M,N>& m, float f);

// Divide matrix by a scalar.
template<unsigned int M, unsigned int N>
Matrix<M,N>& operator/=(Matrix<M,N>& m, float f);

```

HISTORY

Matrix was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableSetMatrix

2.2.3 Ray

NAME

Ray - Ray class

SYNOPSIS

```
#include <optix.h>

namespace optix
{
    class Ray;
}
```

DESCRIPTION

Ray is an encapsulation of a ray mathematical entity. The origin and direction members specify the ray, while the `ray_type` member specifies which closest-hit/any-hit pair will be used when the ray hits a geometry object. The `tmin/tmax` members specify the interval over which the ray is valid.

To avoid numerical range problems, the value `RT_DEFAULT_MAX` can be used to specify an infinite extent.

During C++ compilation, **Ray** is contained within the `optix::` namespace but has global scope during C compilation. **Ray**'s constructors are not available during C compilation.

MEMBERS

```
// The origin of the ray
float3 origin;

// The direction of the ray
float3 direction;

// The ray type associated with this ray
unsigned int ray_type;

// The min and max extents associated with this ray
float tmin;
float tmax;
```

CONSTRUCTORS

```
// Create a Ray with undefined member values
Ray( void );
```

```
// Create a Ray copied from an exemplar
Ray( const Ray &r );

// Create a ray with a specified origin, direction, ray_type, and min/max extents.
// When tmax is not given, it defaults to RT_DEFAULT_MAX.
Ray( float3 origin, float3 direction, unsigned int ray_type,
     float tmin, float tmax = RT_DEFAULT_MAX);
```

FUNCTIONS

```
// Create a ray with a specified origin, direction, ray type, and min/max extents.
Ray make_Ray( float3 origin,
             float3 direction,
             unsigned int ray_type,
             float tmin,
             float tmax );
```

HISTORY

Ray was introduced in *OptiX* 1.0.

SEE ALSO

rtTrace, *rtContextSetRayTypeCount*, *rtMaterialSetAnyHitProgram*, *rtMaterialSetClosestHitProgram*

2.2.4 rtBuffer

NAME

rtBuffer - Declare a reference to a buffer object.

SYNOPSIS

```
#include <optix.h>

rtBuffer<Type, Dim> name;
```

DESCRIPTION

rtBuffer declares a buffer of type **Type** and dimensionality **Dim**. **Dim** must be between 1 and 4 inclusive and defaults to 1 if not specified. The resulting object provides access to buffer data through the `[]` indexing operator, where the index is either unsigned int, uint2, uint3, or uint4 for 1, 2, 3 or 4-dimensional buffers (respectively). This operator can be used to read from or write to the resulting buffer at the specified index.

The named buffer obeys the runtime name lookup semantics as described in *rtVariable*. A compile error will result if the named buffer is not bound to a buffer object, or is bound to a buffer object of the incorrect type or dimension. The behavior of writing to a read-only buffer is undefined. Reading from a write-only buffer is well defined only if a value has been written previously by the same thread.

This declaration must appear at the file scope (not within a function), and will be visible to all RT_PROGRAM and RT_FUNCTION instances within the same compilation unit.

An annotation may be associated with the buffer variable by using the *rtDeclareAnnotation* macro.

HISTORY

rtBuffer was introduced in *OptiX* 1.0.

SEE ALSO

rtDeclareAnnotation, *rtDeclareVariable*, *rtBufferCreate*, *rtTextureSampler*, *rtVariableSetBuffer*

2.2.5 rtObject

NAME

rtObject - Opaque handle to a *OptiX* object.

SYNOPSIS

```
#include <optix.h>

class rtObject;
```

DESCRIPTION

rtObject is an opaque handle to an *OptiX* object of any type. To set or query the variable value, use **rtVariableSetObject** and **rtVariableGetObject**.

Depending on how exactly the variable is used, only certain concrete types may make sense. For example, when used as an argument to **rtTrace**, the variable must be set to any *OptiX* type of **RTgroup**, **RTselector**, **RTgeometrygroup**, or **RTtransform**.

Note that for certain *OptiX* types, there are more specialized handles available to access a variable. For example, to access an *OptiX* object of type **RTtexturesampler**, a handle of type **rtTextureSampler** provides more functionality than one of the generic type **rtObject**.

HISTORY

rtObject was introduced in *OptiX* 1.0.

SEE ALSO

rtVariableSetObject, *rtVariableGetObject*, *rtTrace*, *rtTextureSampler*, *rtBuffer*

2.2.6 rtTextureSampler

NAME

rtTextureSampler - Declares a reference to a texture sampler object.

SYNOPSIS

```
#include <optix.h>

rtTextureSampler<Type, Dim, ReadMode> texref;
```

DESCRIPTION

rtTextureSampler declares a texture of type **Type** and dimensionality **Dim**. **Dim** must be between 1 and 3 inclusive and defaults to 1 if not specified. The resulting object provides access to texture data through the `tex1D`, `tex2D` and `tex3D` functions. These functions can be used only to read the data.

Texture filtering and wrapping modes, specified in **ReadMode** will be dependent on the state of the texture sampler object created with **rtTextureSamplerCreate**.

An annotation may be associated with the texture sampler variable by using the *rtDeclareAnnotation* macro.

HISTORY

rtTextureSampler was introduced in *OptiX* 1.0.

SEE ALSO

rtDeclareAnnotation, *rtTextureSamplerCreate*

2.3 Functions

2.3.1 `rtGetExceptionCode`

NAME

`rtGetExceptionCode` - Retrieves the type of a caught exception.

SYNOPSIS

```
#include <optix.h>

unsigned int rtGetExceptionCode()
```

DESCRIPTION

`rtGetExceptionCode` can be called from an exception program to query which type of exception was caught. The returned code is equivalent to one of the `RTEException` constants passed to `rtContextSetExceptionEnabled`, `RT_EXCEPTION_ALL` excluded. For user-defined exceptions, the code is equivalent to the argument passed to `rtThrow`.

HISTORY

`rtGetExceptionCode` was introduced in *OptiX* 1.1.

SEE ALSO

rtContextSetExceptionEnabled, *rtContextGetExceptionEnabled*, *rtContextSetExceptionProgram*, *rtContextGetExceptionProgram*, *rtThrow*, *rtPrintExceptionDetails*

2.3.2 `rtGetTransform`

NAME

`rtGetTransform` - Get requested transform.

SYNOPSIS

```
#include <optix.h>

void rtGetTransform(RTtransformkind kind,
                   float matrix[16])
```

PARAMETERS

`kind`

The type of transform to retrieve.

`matrix`

Return parameter for the requested transform.

DESCRIPTION

`rtGetTransform` returns the requested transform in the return parameter `matrix`. The type of transform to be retrieved is specified with the `kind` parameter. `kind` is an enumerated value that can be either `RT_OBJECT_TO_WORLD` or `RT_WORLD_TO_OBJECT` and must be a constant literal. During traversal, intersection and any-hit programs, the current ray will be located in object space. During ray generation, closest-hit and miss programs, the current ray will be located in world space.

There may be significant performance overhead associated with a call to `rtGetTransform` compared to a call to `rtTransformPoint`, `rtTransformVector`, or `rtTransformNormal`.

HISTORY

`rtGetTransform` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformPoint*, *rtTransformVector*, *rtTransformNormal*

2.3.3 `rtIgnoreIntersection`

NAME

`rtIgnoreIntersection` - Cancels the potential intersection with current ray.

SYNOPSIS

```
#include <optix.h>

void rtIgnoreIntersection()
```

DESCRIPTION

`rtIgnoreIntersection` causes the current potential intersection to be ignored. This intersection will not become the new closest hit associated with the ray. This function does not return, so values affecting the per-ray data should be applied before calling **`rtIgnoreIntersection`**. **`rtIgnoreIntersection`** is valid only within an any-hit program.

`rtIgnoreIntersection` can be used to implement alpha-mapped transparency by ignoring intersections that hit the geometry but are labeled as transparent in a texture. Since any-hit programs are called frequently during intersection, care should be taken to make them as efficient as possible.

HISTORY

`rtIgnoreIntersection` was introduced in *OptiX* 1.0.

SEE ALSO

`rtTerminateRay`, *`rtPotentialIntersection`*

2.3.4 `rtIntersectChild`

NAME

`rtIntersectChild` - Visit child of selector.

SYNOPSIS

```
#include <optix.h>

void rtIntersectChild(unsigned int index)
```

DESCRIPTION

`rtIntersectChild` will perform intersection on the specified child for the current active ray. This is used in a selector visit program to traverse one of the selector's children. The `index` specifies which of the children to be visited. As the child is traversed, intersection programs will be called and any-hit programs will be called for positive intersections. When this process is complete, `rtIntersectChild` will return unless one of the any-hit programs calls `rtTerminateRay`, in which case this function will never return. Multiple children can be visited during a single selector visit call by calling this function multiple times.

`index` matches the index used in `rtSelectorSetChild` on the host. `rtIntersectChild` is valid only within a selector visit program.

HISTORY

`rtIntersectChild` was introduced in *OptiX* 1.0.

SEE ALSO

rtSelectorSetVisitProgram, *rtSelectorCreate*, *rtTerminateRay*

2.3.5 `rtPotentialIntersection`

NAME

`rtPotentialIntersection` - Determine whether a computed intersection is potentially valid.

SYNOPSIS

```
#include <optix.h>

bool rtPotentialIntersection(float t)
```

DESCRIPTION

Reporting an intersection from a geometry program is a two-stage process. If the geometry program computes that the ray intersects the geometry, it will first call **`rtPotentialIntersection`**. **`rtPotentialIntersection`** will determine whether the reported hit distance is within the valid interval associated with the ray, and return true if the intersection is valid. Subsequently, the geometry program will compute the attributes (normal, texture coordinates, etc.) associated with the intersection before calling **`rtReportIntersection`**. When **`rtReportIntersection`** is called, the any-hit program associated with the material is called. If the any-hit program does not ignore the intersection then the `t` value will stand as the new closest intersection.

If **`rtPotentialIntersection`** returns true, then **`rtReportIntersection`** should **always** be called after computing the attributes. Furthermore, attributes variables should only be written after a successful return from **`rtPotentialIntersection`**.

`rtPotentialIntersection` is passed the material index associated with the reported intersection. Objects with a single material should pass an index of zero.

`rtReportIntersection` and **`rtPotentialIntersection`** are valid only within a geometry intersection program.

HISTORY

`rtPotentialIntersection` was introduced in *OptiX* 1.0.

SEE ALSO

rtGeometrySetIntersectionProgram, *rtReportIntersection*, *rtIgnoreIntersection*

2.3.6 `rtPrintExceptionDetails`

NAME

`rtPrintExceptionDetails` - Print information on a caught exception.

SYNOPSIS

```
#include <optix.h>

void rtPrintExceptionDetails()
```

DESCRIPTION

`rtGetExceptionCode` can be called from an exception program to provide information on the caught exception to the user. The function uses `rtPrintf` to output details depending on the type of the exception. It is necessary to have printing enabled using `rtContextSetPrintEnabled` for this function to have any effect.

HISTORY

`rtPrintExceptionDetails` was introduced in *OptiX* 1.1.

SEE ALSO

rtContextSetExceptionEnabled, *rtContextGetExceptionEnabled*, *rtContextSetExceptionProgram*, *rtContextGetExceptionProgram*, *rtContextSetPrintEnabled*, *rtGetExceptionCode*, *rtThrow*, *rtPrintf*

2.3.7 `rtPrintf`

NAME

rtPrintf - Prints text to the standard output.

SYNOPSIS

```
#include <optix.h>

void rtPrintf(const char* format, ...)
```

DESCRIPTION

rtPrintf is used to output text from within user programs. Arguments are passed as for the standard C **printf** function, and the same format strings are employed. The only exception is the "%s" format specifier, which will generate an error if used. Text printed using **rtPrintf** is accumulated in a buffer and printed to the standard output when **rtContextLaunch** finishes. The buffer size can be configured using **rtContextSetPrintBufferSize**. Output can optionally be restricted to certain launch indices using **rtContextSetPrintLaunchIndex**. Printing must be enabled using **rtContextSetPrintEnabled**, otherwise **rtPrintf** invocations will be silently ignored.

HISTORY

rtPrintf was introduced in *OptiX* 1.0.

SEE ALSO

rtContextSetPrintEnabled, *rtContextGetPrintEnabled*, *rtContextSetPrintBufferSize*, *rtContextGetPrintBufferSize*, *rtContextSetPrintLaunchIndex*, *rtContextSetPrintLaunchIndex*

2.3.8 `rtReportIntersection`

NAME

`rtReportIntersection` - Report an intersection with the current object and the specified material.

SYNOPSIS

```
#include <optix.h>

bool rtReportIntersection(unsigned int material)
```

DESCRIPTION

`rtReportIntersection` reports an intersection of the current ray with the current object, and specifies the material associated with the intersection. **`rtReportIntersection`** should only be used in conjunction with **`rtPotentialIntersection`** as described in **`rtPotentialIntersection`**.

HISTORY

`rtReportIntersection` was introduced in *OptiX* 1.0.

SEE ALSO

`rtPotentialIntersection`

2.3.9 `rtTerminateRay`

NAME

`rtTerminateRay` - Terminate traversal associated with the current ray.

SYNOPSIS

```
#include <optix.h>

void rtTerminateRay()
```

DESCRIPTION

`rtTerminateRay` causes the traversal associated with the current ray to immediately terminate. After termination, the closest-hit program associated with the ray will be called. This function does not return, so values affecting the per-ray data should be applied before calling `rtTerminateRay`. `rtTerminateRay` is valid only within an any-hit program.

This function can be used to provide early-ray termination for opaque objects in shadow rays.

HISTORY

`rtTerminateRay` was introduced in *OptiX* 1.0.

SEE ALSO

rtPotentialIntersection, *rtIgnoreIntersection*

2.3.10 `rtThrow`

NAME

`rtThrow` - Throw a user exception.

SYNOPSIS

```
#include <optix.h>

void rtThrow(unsigned int code)
```

DESCRIPTION

`rtThrow` is used to trigger user defined exceptions which behave like built-in exceptions. That is, upon invocation, ray processing for the current launch index is immediately aborted and the corresponding exception program is executed. `rtThrow` does not return.

The `code` passed as argument must be within the range reserved for user exceptions, which starts at `RT_EXCEPTION_USER` (0x400) and ends at 0xFFFF. The code can be queried within the exception program using `rtGetExceptionCode`.

`rtThrow` may be called from within any program type except exception programs. Calls to `rtThrow` will be silently ignored unless user exceptions are enabled using `rtContextSetExceptionEnabled`.

HISTORY

`rtThrow` was introduced in *OptiX* 1.1.

SEE ALSO

rtContextSetExceptionEnabled, *rtContextGetExceptionEnabled*, *rtContextSetExceptionProgram*, *rtContextGetExceptionProgram*, *rtGetExceptionCode*, *rtPrintExceptionDetails*

2.3.11 `rtTrace`

NAME

`rtTrace` - Traces a ray.

SYNOPSIS

```
#include <optix.h>

void rtTrace(rtObject topNode,
             Ray ray,
             T& prd)
```

DESCRIPTION

`rtTrace` traces `ray` against object `topNode`. A reference to `prd`, the per-ray data, will be passed to all of the closest-hit and any-hit programs that are executed during this invocation of trace. `topNode` must refer to an *OptiX* object of type `RTgroup`, `RTselector`, `RTgeometrygroup`, or `RTtransform`.

HISTORY

`rtTrace` was introduced in *OptiX* 1.0.

SEE ALSO

rtObject, *Ray*

2.3.12 `rtTransformNormal`

NAME

`rtTransformNormal` - Apply the current transformation to a normal.

SYNOPSIS

```
#include <optix.h>

float3 rtTransformNormal(RTtransformkind kind,
                        const float3& n)
```

DESCRIPTION

`rtTransformNormal` transforms `n` as a normal using the current active transformation stack (the inverse transpose). During traversal, intersection and any-hit programs, the current ray will be located in object space. During ray generation, closest-hit and miss programs, the current ray will be located in world space. This function can be used to transform values between object and world space.

`kind` is an enumerated value that can be either `RT_OBJECT_TO_WORLD` or `RT_WORLD_TO_OBJECT` and must be a constant literal. For ray generation and miss programs, the transform will always be the identity transform. For traversal, intersection, any-hit and closest-hit programs, the transform will be dependent on the set of active transform nodes for the current state.

HISTORY

`rtTransformNormal` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformPoint*, *rtTransformVector*

2.3.13 `rtTransformPoint`

NAME

`rtTransformPoint` - Apply the current transformation to a point.

SYNOPSIS

```
#include <optix.h>

float3 rtTransformPoint(RTtransformkind kind,
                       const float3& p)
```

DESCRIPTION

`rtTransformPoint` transforms `p` as a point using the current active transformation stack. During traversal, intersection and any-hit programs, the current ray will be located in object space. During ray generation, closest-hit and miss programs, the current ray will be located in world space. This function can be used to transform the ray origin and other points between object and world space.

`kind` is an enumerated value that can be either `RT_OBJECT_TO_WORLD` or `RT_WORLD_TO_OBJECT` and must be a constant literal. For ray generation and miss programs, the transform will always be the identity transform. For traversal, intersection, any-hit and closest-hit programs, the transform will be dependent on the set of active transform nodes for the current state.

HISTORY

`rtTransformPoint` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformVector*, *rtTransformNormal*

2.3.14 `rtTransformVector`

NAME

`rtTransformVector` - Apply the current transformation to a vector.

SYNOPSIS

```
#include <optix.h>

float3 rtTransformVector(RTtransformkind kind,
                        const float3& v)
```

DESCRIPTION

`rtTransformVector` transforms `v` as a vector using the current active transformation stack. During traversal, intersection and any-hit programs, the current ray will be located in object space. During ray generation, closest-hit and miss programs, the current ray will be located in world space. This function can be used to transform the ray direction and other vectors between object and world space.

`kind` is an enumerated value that can be either `RT_OBJECT_TO_WORLD` or `RT_WORLD_TO_OBJECT` and must be a constant literal. For ray generation and miss programs, the transform will always be the identity transform. For traversal, intersection, any-hit and closest-hit programs, the transform will be dependent on the set of active transform nodes for the current state.

HISTORY

`rtTransformVector` was introduced in *OptiX* 1.0.

SEE ALSO

rtTransformCreate, *rtTransformVector*, *rtTransformNormal*

Chapter 3

Appendix

3.1 Interop Formats

DESCRIPTION

This section lists OpenGL and D3D texture formats that are currently supported for interop:

OpenGL

*R8I, R8UI, RG8I, RG8UI, RGBA8, RGBA8I, RGBA8UI,
R16I, R16UI, RG16I, RG16UI, RGBA16, RGBA16I, RGBA16UI,
R32I, R32UI, RG32I, RG32UI, RGBA32I, RGBA32UI,
R32F, RG32F, RGBA32F*

D3D Formats

*L8, A8
A8L8, V8U8
A8R8G8B8, X8R8G8B8, A8B8G8R8, X8B8G8R8, Q8W8V8U8
L16, G16R16, V16U16
A16B16G16R16, Q16W16V16U16
R32F, G32R32F, A32B32G32R32F*

DXGI Formats

*R8_SINT, R8_SNORM, R8_UINT, R8_UNORM
R16_SINT, R16_SNORM, R16_UINT, R16_UNORM
R32_SINT, R32_UINT, R32_FLOAT
R8G8_SINT, R8G8_SNORM, R8G8_UINT, R8G8_UNORM
R16G16_SINT, R16G16_SNORM, R16G16_UINT, R16G16_UNORM
R32G32_SINT, R32G32_UINT, R32G32_FLOAT
R8G8B8A8_SINT, R8G8B8A8_SNORM, R8G8B8A8_UINT, R8G8B8A8_UNORM
R16G16B16A16_SINT, R16G16B16A16_SNORM, R16G16B16A16_UINT, R16G16B16A16_UNORM
R32G32B32A32_SINT, R32G32B32A32_UINT, R32G32B32A32_FLOAT*

HISTORY

Interop Texture Formats were introduced in *OptiX 2.0*.

SEE ALSO

rtTextureSamplerCreateFromGLImage

rtTextureSamplerCreateFromD3D9Resource

rtTextureSamplerCreateFromD3D10Resource

rtTextureSamplerCreateFromD3D11Resource

Index

- Aabb, 386
- Acceleration Structure, 129
- API Reference, 1
- Appendix, 413

- Buffer, 239

- Context, 2
- Context Free Functions, 365
- CUDA C Reference, 377

- Declarations, 378

- Functions, 397

- Geometry, 176
- Geometry Group, 62
- Geometry Instance, 152
- Group Node, 78

- Interop Formats, 414

- Material, 200
- Matrix, 389

- Program, 223

- Ray, 392
- RT FUNCTION, 383
- RT PROGRAM, 384
- rtAccelerationCreate, 130
- rtAccelerationDestroy, 132
- rtAccelerationGetBuilder, 133
- rtAccelerationGetContext, 134
- rtAccelerationGetData, 135
- rtAccelerationGetDataSize, 136
- rtAccelerationGetProperty, 137
- rtAccelerationGetTraverser, 139
- rtAccelerationIsDirty, 140
- rtAccelerationMarkDirty, 142
- rtAccelerationSetBuilder, 143
- rtAccelerationSetData, 145
- rtAccelerationSetProperty, 147
- rtAccelerationSetTraverser, 149
- rtAccelerationValidate, 151
- rtBuffer, 394
- rtBufferCreate, 241
- rtBufferCreateFromD3D10Resource, 247
- rtBufferCreateFromD3D11Resource, 249
- rtBufferCreateFromD3D9Resource, 245
- rtBufferCreateFromGLBO, 243
- rtBufferD3D10Register, 255
- rtBufferD3D10Unregister, 252
- rtBufferD3D11Register, 256
- rtBufferD3D11Unregister, 253
- rtBufferD3D9Register, 254
- rtBufferD3D9Unregister, 251
- rtBufferDestroy, 257
- rtBufferGetContext, 258
- rtBufferGetD3D10Resource, 263
- rtBufferGetD3D11Resource, 264
- rtBufferGetD3D9Resource, 262
- rtBufferGetDimensionality, 259
- rtBufferGetElementSize, 260
- rtBufferGetFormat, 261
- rtBufferGetGLBOId, 265
- rtBufferGetSize1D, 266
- rtBufferGetSize2D, 267
- rtBufferGetSize3D, 269
- rtBufferGetSizev, 271
- rtBufferGLRegister, 274
- rtBufferGLUnregister, 273
- rtBufferMap, 275
- rtBufferSetElementSize, 277
- rtBufferSetSize1D, 279
- rtBufferSetSize2D, 280
- rtBufferSetSize3D, 282
- rtBufferSetSizev, 284
- rtBufferUnmap, 286
- rtBufferValidate, 287
- rtContextCompile, 4
- rtContextCreate, 6
- rtContextDeclareVariable, 7
- rtContextDestroy, 9
- rtContextGetAttribute, 10
- rtContextGetDeviceCount, 13

- rtContextGetDevices, 12
- rtContextGetEntryPointCount, 14
- rtContextGetErrorString, 15
- rtContextGetExceptionEnabled, 16
- rtContextGetExceptionProgram, 18
- rtContextGetMissProgram, 20
- rtContextGetPrintBufferSize, 21
- rtContextGetPrintEnabled, 22
- rtContextGetPrintLaunchIndex, 23
- rtContextGetRayGenerationProgram, 25
- rtContextGetRayTypeCount, 27
- rtContextGetRunningState, 28
- rtContextGetStackSize, 29
- rtContextGetVariable, 31
- rtContextGetVariableCount, 30
- rtContextLaunch, 59
- rtContextQueryVariable, 33
- rtContextRemoveVariable, 35
- rtContextSetAttribute, 36
- rtContextSetD3D10Device, 39
- rtContextSetD3D11Device, 40
- rtContextSetD3D9Device, 38
- rtContextSetDevices, 41
- rtContextSetEntryPointCount, 42
- rtContextSetExceptionEnabled, 43
- rtContextSetExceptionProgram, 45
- rtContextSetMissProgram, 47
- rtContextSetPrintBufferSize, 49
- rtContextSetPrintEnabled, 50
- rtContextSetPrintLaunchIndex, 51
- rtContextSetRayGenerationProgram, 53
- rtContextSetRayTypeCount, 55
- rtContextSetStackSize, 56
- rtContextSetTimeoutCallback, 57
- rtContextValidate, 61
- rtDeclareAnnotation, 379
- rtDeclareVariable, 381
- rtDeviceGetAttribute, 366
- rtDeviceGetD3D10Device, 371
- rtDeviceGetD3D11Device, 373
- rtDeviceGetD3D9Device, 369
- rtDeviceGetDeviceCount, 368
- rtDeviceGetWGLDevice, 375
- rtGeometryCreate, 177
- rtGeometryDeclareVariable, 178
- rtGeometryDestroy, 180
- rtGeometryGetBoundingBoxProgram, 181
- rtGeometryGetContext, 182
- rtGeometryGetIntersectionProgram, 183
- rtGeometryGetPrimitiveCount, 184
- rtGeometryGetVariable, 186
- rtGeometryGetVariableCount, 185
- rtGeometryGroupCreate, 63
- rtGeometryGroupDestroy, 64
- rtGeometryGroupGetAcceleration, 65
- rtGeometryGroupGetChild, 68
- rtGeometryGroupGetChildCount, 66
- rtGeometryGroupGetContext, 70
- rtGeometryGroupSetAcceleration, 71
- rtGeometryGroupSetChild, 75
- rtGeometryGroupSetChildCount, 73
- rtGeometryGroupValidate, 77
- rtGeometryInstanceCreate, 153
- rtGeometryInstanceDeclareVariable, 154
- rtGeometryInstanceDestroy, 156
- rtGeometryInstanceGetContext, 157
- rtGeometryInstanceGetGeometry, 158
- rtGeometryInstanceGetMaterial, 160
- rtGeometryInstanceGetMaterialCount, 159
- rtGeometryInstanceGetVariable, 164
- rtGeometryInstanceGetVariableCount, 162
- rtGeometryInstanceQueryVariable, 166
- rtGeometryInstanceRemoveVariable, 168
- rtGeometryInstanceSetGeometry, 170
- rtGeometryInstanceSetMaterial, 173
- rtGeometryInstanceSetMaterialCount, 171
- rtGeometryInstanceValidate, 175
- rtGeometryIsDirty, 188
- rtGeometryMarkDirty, 190
- rtGeometryQueryVariable, 191
- rtGeometryRemoveVariable, 193
- rtGeometrySetBoundingBoxProgram, 195
- rtGeometrySetIntersectionProgram, 196
- rtGeometrySetPrimitiveCount, 198
- rtGeometryValidate, 199
- rtGetExceptionCode, 398
- rtGetTransform, 399
- rtGetVersion, 376
- rtGroupCreate, 79
- rtGroupDestroy, 80
- rtGroupGetAcceleration, 81
- rtGroupGetChild, 83
- rtGroupGetChildCount, 82
- rtGroupGetChildType, 84
- rtGroupGetContext, 85
- rtGroupSetAcceleration, 86
- rtGroupSetChild, 89
- rtGroupSetChildCount, 88
- rtGroupValidate, 91
- rtIgnoreIntersection, 400
- rtIntersectChild, 401
- rtMaterialCreate, 201

- rtMaterialDeclareVariable, 203
- rtMaterialDestroy, 205
- rtMaterialGetAnyHitProgram, 206
- rtMaterialGetClosestHitProgram, 208
- rtMaterialGetContext, 210
- rtMaterialGetVariable, 212
- rtMaterialGetVariableCount, 211
- rtMaterialQueryVariable, 214
- rtMaterialRemoveVariable, 216
- rtMaterialSetAnyHitProgram, 218
- rtMaterialSetClosestHitProgram, 220
- rtMaterialValidate, 222
- rtObject, 395
- rtPotentialIntersection, 402
- rtPrintExceptionDetails, 403
- rtPrintf, 404
- rtProgramCreateFromPTXFile, 224
- rtProgramCreateFromPTXString, 226
- rtProgramDeclareVariable, 228
- rtProgramDestroy, 230
- rtProgramGetContext, 231
- rtProgramGetVariable, 233
- rtProgramGetVariableCount, 232
- rtProgramQueryVariable, 235
- rtProgramRemoveVariable, 237
- rtProgramValidate, 238
- rtReportIntersection, 405
- rtSelectorCreate, 93
- rtSelectorDeclareVariable, 94
- rtSelectorDestroy, 96
- rtSelectorGetChild, 98
- rtSelectorGetChildCount, 97
- rtSelectorGetChildType, 100
- rtSelectorGetContext, 102
- rtSelectorGetVariable, 104
- rtSelectorGetVariableCount, 103
- rtSelectorGetVisitProgram, 106
- rtSelectorQueryVariable, 107
- rtSelectorRemoveVariable, 109
- rtSelectorSetChild, 111
- rtSelectorSetChildCount, 110
- rtSelectorSetVisitProgram, 113
- rtSelectorValidate, 115
- rtTerminateRay, 406
- rtTextureSampler, 396
- rtTextureSamplerCreate, 290
- rtTextureSamplerCreateFromD3D10Resource, 295
- rtTextureSamplerCreateFromD3D11Resource, 297
- rtTextureSamplerCreateFromD3D9Resource, 293
- rtTextureSamplerCreateFromGLImage, 291
- rtTextureSamplerD3D10Register, 303
- rtTextureSamplerD3D10Unregister, 300
- rtTextureSamplerD3D11Register, 304
- rtTextureSamplerD3D11Unregister, 301
- rtTextureSamplerD3D9Register, 302
- rtTextureSamplerD3D9Unregister, 299
- rtTextureSamplerDestroy, 305
- rtTextureSamplerGetArraySize, 306
- rtTextureSamplerGetBuffer, 307
- rtTextureSamplerGetContext, 309
- rtTextureSamplerGetD3D10Resource, 311
- rtTextureSamplerGetD3D11Resource, 312
- rtTextureSamplerGetD3D9Resource, 310
- rtTextureSamplerGetFilteringModes, 313
- rtTextureSamplerGetGLImageId, 315
- rtTextureSamplerGetIndexingMode, 317
- rtTextureSamplerGetMaxAnisotropy, 318
- rtTextureSamplerGetMipLevelCount, 319
- rtTextureSamplerGetReadMode, 320
- rtTextureSamplerGetWrapMode, 321
- rtTextureSamplerGLRegister, 323
- rtTextureSamplerGLUnregister, 322
- rtTextureSamplerSetArraySize, 324
- rtTextureSamplerSetBuffer, 325
- rtTextureSamplerSetFilteringModes, 327
- rtTextureSamplerSetIndexingMode, 329
- rtTextureSamplerSetMaxAnisotropy, 331
- rtTextureSamplerSetMipLevelCount, 332
- rtTextureSamplerSetReadMode, 333
- rtTextureSamplerSetWrapMode, 335
- rtTextureSamplerValidate, 337
- rtThrow, 407
- rtTrace, 408
- rtTransformCreate, 117
- rtTransformDestroy, 118
- rtTransformGetChild, 119
- rtTransformGetChildType, 120
- rtTransformGetContext, 122
- rtTransformGetMatrix, 123
- rtTransformNormal, 409
- rtTransformPoint, 410
- rtTransformSetChild, 125
- rtTransformSetMatrix, 126
- rtTransformValidate, 128
- rtTransformVector, 411
- rtVariableGet, 339
- rtVariableGetAnnotation, 347
- rtVariableGetContext, 344
- rtVariableGetName, 345
- rtVariableGetObject, 349
- rtVariableGetSize, 351
- rtVariableGetType, 352

rtVariableGetUserData, [354](#)

rtVariableSet, [356](#)

rtVariableSetObject, [361](#)

rtVariableSetUserData, [363](#)

SEE ALSO [rtContextGetDevices](#), [rtContextGetDeviceCount](#), [41](#)

Selector Node, [92](#)

Texture Sampler, [288](#)

Transform Node, [116](#)

Types, [385](#)

Variables, [338](#)