RayBooster 4 documentation

Jean-Patrick ROCCIA

August 2013
## Contents

1 Overview .......................................................... 3
   1.1 What is RayBooster? ........................................... 3
   1.2 Requirements .................................................. 3
   1.3 Conventions .................................................... 3
      1.3.1 Syntax .................................................... 3
      1.3.2 Return values ........................................... 4
      1.3.3 Identifiers ............................................... 4

2 Concepts .......................................................... 4
   2.1 Driver .......................................................... 4
   2.2 Scene .......................................................... 4
   2.3 Geometry ....................................................... 5
   2.4 Instances ....................................................... 5
   2.5 Layers .......................................................... 5
   2.6 Ray tracing ..................................................... 5
   2.7 Work-flow ..................................................... 5
      2.7.1 Idle stage ............................................... 5
      2.7.2 Driver stage ............................................. 6
      2.7.3 Scene stage ............................................. 6
      2.7.4 Ray tracing stage ...................................... 6
      2.7.5 Ray tracing session .................................... 6

3 API .................................................................. 6
   3.1 Types ............................................................ 6
   3.2 User data access ............................................... 7
   3.3 Entry points ..................................................... 7
      3.3.1 Drives ...................................................... 7
      3.3.2 Scene ........................................................ 11
      3.3.3 Geometry .................................................. 16
      3.3.4 Instance .................................................... 20
      3.3.5 Ray Tracing .............................................. 30

4 Threading .......................................................... 38
1 Overview

This document describes the RayBooster 4 Ray Tracing Acceleration Library. It is assumed that the reader has notions of programming in C or C++ languages, as well as notions of Ray Tracing and geometrics.

1.1 What is RayBooster?

RayBooster 4 is a Ray Tracing Acceleration Library. It provides software functions and utilities in order to handle geometric and intersection requests in an easy, non-intrusive and efficient way. As you can see in the figure 1, RayBooster 4 handles all ray tracing specific acceleration datas and intersection tests, but the physics used to found intersections remains on the host side.

![Ray Tracing Process Diagram]

Figure 1: This schema describes a typical and general ray tracing process. Blue cells represent steps operated by RayBooster 4, which are independent of the simulation domain. Red cells remain on the user side because of their strong specific simulation domain dependency.

1.2 Requirements

RayBooster 4 is a C++ library. Being essentially a mathematical library, it doesn’t need any compatibility requirement such as OpenGL, DirectX or any other. System specific functions are used, particularly for multi-threading purpose and advanced low-level optimization. Anyway, these calls are transparents and shall not perturb the host application.

On the hardware side, the requirements depend of the targeted RayBooster 4 driver. Currently, the only one released driver require an x86 processor with SSE2 instructions enabled (at least Intel Pentium 4 or AMD Athlon 64).

Additionally, RayBooster 4 is available on the following platforms, in both 32 bits and 64 bits:

- Microsoft Windows
- Apple MAC OS
- Linux

1.3 Conventions

1.3.1 Syntax

The RayBooster 4 interface obeys to the following syntax:
• Structures : SRBStructureName
• Structures members : m_MemberName
• Functions : RBFUNCTIONNAME
• Enumeration types : ERBEnumerationType
• Enumeration members : ERBEnumerationType_MemberName
• Macros/Compilation flags : RB_MACRO

1.3.2 Return values
All RayBooster 4 functions return an "enum ERBResult" result code. Exact signification of the return code depends on the function itself and is described in its documentation. All error codes and their rough descriptions are available in the table 1.

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Everything went well.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>Incorrect input data provided by user.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>Internal memory allocation or deallocation went bad.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Internal Input/Output access went bad.</td>
</tr>
</tbody>
</table>

Table 1: List of all result code values and their rough description.

1.3.3 Identifiers
All interactions with the RayBooster 4 library go through identifiers. Only the RayBooster 4 driver is directly manipulated by users. It means that all objects are internally managed by the driver, and the only way to impact them is to use the driver entry points with their identifiers.

2 Concepts
RayBooster 4 relies on a set of concepts leading to a natural and easy to use work-flow. Each concept and its associated work-flow is described below.

2.1 Driver
A driver is a specific implementation of the RayBooster 4 Application Programming Interface (API). Actually the driver is loaded dynamically according to detected hardware and its capacities. The driver contains all entry points of the RayBooster 4 API. Many drivers can work in parallel but they don't share any data or identifiers. RayBooster 4 drivers have capacities to create and manage scenes.

2.2 Scene
A scene is a set of instantiated geometries. Each ray tracing request is applied to a scene, so the scene represent a set of geometries that can be intersected for a given request. A scene is strongly linked to the driver that created it. Indeed, the driver is the only interface between user and scene.
2.3 Geometry

A geometry, in the meaning of RayBooster 4, is an access to the client triangulated geometry structure. A geometry can be registered in scenes, in order to be instantiated in them. Adding a geometry in a scene doesn’t make it available for the ray tracing. The geometry must be instantiated in order to be intersected.

2.4 Instances

An instance is the association of a geometry and a transformation matrix. A geometry can be instantiated several times in a scene, with different matrices. Instantiating similar geometries allows to reduce memory consumption, by not duplicating geometrical data. Instantiating a geometry in a scene makes it available for the ray tracing.

2.5 Layers

Layers offer the possibility to partition geometric instances inside a scene. An instance can be declared active for only a sub-set of layers. RayBooster 4 allows up to thirty-two distinct layers, and each call to the ray tracing can be performed on a sub-set of those layers.

2.6 Ray tracing

This part is the core of RayBooster 4. Users provide rays that must be intersected with a given scene, and gets back intersections with geometries. For each set of rays to intersect, users can specify a set of active layers.

2.7 Work-flow

The RayBooster 4 work-flow can be divided into five specific stages, as shown in the figure 2. Each stage will be described more precisely in this section.

Figure 2: This schema describes the different RayBooster 4 work flow stages. Each stage provides its own features which are described below.

2.7.1 Idle stage

This stage is the initial one for users. The only one feature provided at this step is RayBooster 4 driver loading. A driver is required for any interaction with RayBooster 4. Loading a driver allows to switch to the driver stage.
2.7.2 Driver stage

At this stage, the driver offers the possibility to create a RayBooster 4 scene. In order to switch back to the previous stage, it is possible to unload the driver. Creating a scene allows to switch to the scene stage.

2.7.3 Scene stage

The scene stage offers features in order to describe the client geometry. It is possible to:

- add/remove a geometry
- add/remove instance of a registered geometry
- set geometric transformations of a registered instance
- set active layers of a registered instance

Locking the scene switches to the ray tracing stage while scene destruction switches back to the driver stage.

2.7.4 Ray tracing stage

At this stage, RayBooster 4 reject all geometric modifications on the locked scene. The only one step required to start a ray tracing session is the scene data building. This construction is very important because it makes all previous scene modifications effective. After completing this construction step, a valid ray tracing session becomes active, while unlocking the scene switches back to the scene stage.

2.7.5 Ray tracing session

At this point RayBooster 4 offers the possibilities to ray trace an isolated ray, or a set of rays. Moreover, it is possible to obtain the Aligned on Axis Bounding Box (AABB) of the scene or the AABB of a given set of layers. Unlocking the scene switches back to the scene stage.

3 API

The RayBooster 4 Application Programming Interface implements all previously introduced concepts. The API uses types defined in an independent project: RBUtils. The RBUtils library defines a lot of system and mathematical primitives, which are used by the RayBooster 4 API and can be used as well by RayBooster 4 users.

3.1 Types

In this section, all types required by API function call are shortly described. First, general utilities types in the table 2, followed by Ray Tracing specific types in the table 3.
Table 2: Short description of all RBUtils types appearing in the RayBooster 4 API.

<table>
<thead>
<tr>
<th>Type name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRBFloat2</td>
<td>Contains two float values.</td>
</tr>
<tr>
<td>SRBFloat3</td>
<td>Contains three float values.</td>
</tr>
<tr>
<td>SRBFloat44</td>
<td>Contains a matrix of 4x4 float values.</td>
</tr>
<tr>
<td>SRBBox</td>
<td>Parallelepiped box aligned on axis (AABB).</td>
</tr>
</tbody>
</table>

Table 3: Short description of all RBAPI types appearing in the RayBooster 4 API.

<table>
<thead>
<tr>
<th>Type name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBDriverType</td>
<td>Enumeration of loadable types of drivers.</td>
</tr>
<tr>
<td>IRBDriver</td>
<td>Contains all API entry points.</td>
</tr>
<tr>
<td>RBSceneId</td>
<td>Unique scene identifier in a given driver.</td>
</tr>
<tr>
<td>RBGeomId</td>
<td>Unique geometry identifier in a given scene.</td>
</tr>
<tr>
<td>RBInstanceId</td>
<td>Unique geometric instance identifier in a given scene.</td>
</tr>
<tr>
<td>RBLayerId</td>
<td>Composition of layers identifiers.</td>
</tr>
<tr>
<td>SRBPrimitiveId</td>
<td>Unique triangle identifier in a given scene.</td>
</tr>
<tr>
<td>SRBTriangulatedGeomDescriptor</td>
<td>Contains callbacks for user geometry access.</td>
</tr>
<tr>
<td>RBTraceOptions</td>
<td>Describes inputs ray tracing data.</td>
</tr>
<tr>
<td>SRBRay</td>
<td>Contains the complete description of a ray.</td>
</tr>
<tr>
<td>SRBHit</td>
<td>Contains results of a ray/scene intersection.</td>
</tr>
</tbody>
</table>

3.2 User data access

RayBooster 4 uses an ingenious mechanism of callbacks in order to access user geometry in a non-intrusive way while avoiding memory duplication. This mechanism is symbolized by the structure SRBTriangulatedGeomDescriptor. In this structure, users can specify a pointer on their data type and specify two methods:

- A method returning the number of triangles of this geometry, using or not the pointer on user data.
- A method returning the requested triangle, using or not the pointer on user data.

RayBooster 4 will use those methods for all user geometry access, without imposing a geometry structure or a memory layout to users.

3.3 Entry points

In this section, all exposed functions in the RayBooster 4 API are precisely described, according to their purpose.

3.3.1 Drivers
enum ERBResult RBDriverLoad
   ( enum ERBDriverType DriverType, struct IRBDriver* Driver );

**Description**
Try to load the most accurate RayBooster 4 driver, according to the user requested driver type.

**Parameters**
*DriverType*
Input user driver type picked in the enumeration ERBDriverType, described in the table 4.

<table>
<thead>
<tr>
<th>Driver type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBDriverCpu</td>
<td>Best CPU driver according to detected hardware.</td>
</tr>
<tr>
<td>ERBDriverGpu</td>
<td>Best GPU driver according to detected hardware.</td>
</tr>
<tr>
<td>ERBDriverHybrid</td>
<td>Hybrid CPU/GPU driver.</td>
</tr>
<tr>
<td>ERBDriverAuto</td>
<td>Best driver according to detected hardware.</td>
</tr>
</tbody>
</table>

Table 4: List of all loadable RayBooster 4 driver types.

*Driver*
Pointer on an empty driver interface, allocated by user and filled by the function.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Driver was successfully loaded.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>DriverType is incorrect or Driver is NULL.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>The selected driver seems to be damaged or incompatible.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>The selected driver was not found in the execution path.</td>
</tr>
</tbody>
</table>

Table 5: List of all RBDriverLoad result code values and their description.

**Example**

```c
struct IRBDriver driver;
enum ERBResult RResult = RBDriverLoad(ERBDriverCpu, &driver);
```
enum ERBResult RBDriverLoadSpecific
    ( const RBChar* DriverName, struct IRBDriver* Driver );

**Description**
Try to load the specified RayBooster 4 driver, according to the user requested specific driver name.

**Parameters**
*DriverName*
Input user driver name.

*Driver*
Pointer on an empty driver interface, allocated by user and filled by the function.

**Helper macro**
*RB_DRIVER_TO_LIBNAME*
Build the system dependant library filename from the driver name.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Driver was successfully loaded.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td><em>DriverName</em> and/or <em>Driver</em> is NULL.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>The selected driver seems to be damaged or incompatible.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>The selected driver was not found in the execution path.</td>
</tr>
</tbody>
</table>

Table 6: List of all *RBDriverLoadSpecific* result code values, and their description.

**Example**

```c
struct IRBDriver driver;
enum ERBResult RResult = RBDriverLoadSpecific
    (RB_DRIVER_TO_LIBNAME("RBSe2"), &driver);
```
enum ERBResult RBDriverUnload
    ( struct IRBDriver* Driver );

Description
Try to unload the specified RayBooster 4 driver.

Parameters
Driver
Pointer on a previously loaded driver interface.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Driver was successfully unloaded.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>Driver is NULL or not loaded.</td>
</tr>
</tbody>
</table>

Table 7: List of all RBDriverUnload result code values, and their description.

Example

```
enum ERBResult RBResult = RBDriverUnload(&driver);
```
3.3.2 Scene

```c
enum ERBResult SceneCreate
    ( struct RBSceneId* SceneId );
```

**Description**
Create a RayBooster 4 scene inside a driver.

**Parameters**

SceneId
Pointer to a RBSceneId that will be filled with the newly created unique scene identifier.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully created.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>Unable to allocate a new scene.</td>
</tr>
</tbody>
</table>

Table 8: List of all SceneCreate result code values and their description.

**Example**

```c
RBSceneId sceneId;
enum ERBResult RBResult = driver.SceneCreate(&sceneId);
```
enum ERBResult SceneDestroy
    ( const RBSceneId SceneId );

Description
Destroy a RayBooster 4 scene inside a driver.

Parameters
SceneId
RBSceneId of the scene to destroy.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully destroyed.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid.</td>
</tr>
</tbody>
</table>

Table 9: List of all SceneDestroy result code values and their description.

Example

enum ERBResult RBResult = driver.SceneDestroy(sceneId);
enum ERBResult  SceneClear
    (  const RBSceneId SceneId  );

Description
Reinitialize a RayBooster 4 scene inside a driver, removing all declared geometries and instances.

Parameters
SceneId
RBSceneId to reinitialize.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully cleared.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is currently locked.</td>
</tr>
</tbody>
</table>

Table 10: List of all SceneClear result code values and their description.

Example

eenum ERBResult  RBResult = driver.SceneClear(sceneId);
enum ERBResult SceneLock
    ( const RBSceneId SceneId );

Description
Lock a RayBooster 4 scene inside a driver. All geometries and instances modifications will be
rejected on a locked scene.

Parameters
SceneId
RBSceneId to lock.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully locked.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is already locked.</td>
</tr>
</tbody>
</table>

Table 11: List of all SceneLock result code values and their description.

Example

enum ERBResult RBResult = driver.SceneLock(sceneId);
enum ERBResult SceneUnlock
    ( const RBSceneId SceneId );

Description
Unlock a RayBooster 4 scene inside a driver. All geometries and instances modifications are
allowed again on the scene.

Parameters
SceneId
RBSceneId to unlock.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully unlocked.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is not locked.</td>
</tr>
</tbody>
</table>

Table 12: List of all SceneUnlock result code values and their description.

Example

enum ERBResult RBResult = driver.SceneUnlock(sceneId);
3.3.3 Geometry

```c
enum ERBResult SceneGeometryAdd(
    const RBSceneId SceneId,
    const struct SRB TriangulatedGeomDescriptor
    TriangulatedGeomDescriptor,
    RBGeomId* GeomId
);
```

**Description**
Register a user’s geometry in a RayBooster 4 scene.

**Parameters**

- **SceneId**
  RBSceneId receiving the geometry.

- **TriangulatedGeomDescriptor**
  Descriptor of the user geometry for RayBooster 4 non-intrusive access.

- **GeomId**
  Pointer on a RBGeomId that will be filled with the newly created unique geometry identifier.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Geometry was successfully added.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid or GeomId is NULL.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>No more geometry unique identifier available.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 13: List of all SceneGeometryAdd result code values and their description.

**Example**

```c
RBGeomId geomId = RBINvalidId;
enum ERBResult RBRresult =
    driver.SceneGeometryAdd(sceneId, geomDescriptor, &geomId);
```
enum ERBResult SceneGeometryRemove(
    const RBSceneId SceneId,
    const RBGeomId GeomId);

Description
Remove a user's geometry in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the geometry.

GeomId
RBGeomId to remove.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Geometry was successfully removed.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or GeomId is not valid.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>GeomId is still instantiated.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 14: List of all SceneGeometryRemove result code values and their description.

Example

RBGeomId geomId = RBInvalidId;
enum ERBResult RBResult = driver.SceneGeometryRemove
    (sceneId, geomId);
enum ERBResult SceneGeometryClientDataSet(
    const RBSceneId SceneId,
    const RBGeomId GeomId,
    void * ClientData);

Description
Associate a user data pointer to a registered geometry in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the geometry.

GeomId
RBGeomId identifying the geometry.

ClientData
Pointer on the user data to associate.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>User data pointer was successfully affected.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or GeomId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 15: List of all SceneGeometryClientDataSet result code values, and their description.

Example

void * clientData = (void *)&myData;
enum ERBResult RBResult =
    driver.SceneGeometryClientDataSet(sceneId, geomId, clientData);
enum ERBResult SceneGeometryClientIdaDataGet(
    const RBSceneId SceneId,
    const RBGeomId GeomId,
    void ** ClientData);

Description
Retrieve user data pointer associated to a registered geometry in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the geometry.

GeomId
RBGeomId identifying the geometry associated to the required data pointer.

ClientData
Pointer to fill with the user data pointer.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>User data pointer was successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or GeomId is not valid or ClientData is NULL.</td>
</tr>
</tbody>
</table>

Table 16: List of all SceneGeometryClientIdaDataGet result code values, and their description.

Example

void * clientData = NULL;
enum ERBResult RBResult =
    driver.SceneGeometryClientIdaDataGet(SceneId, GeomId, &clientData);
3.3.4 Instance

```c
enum ERBResult SceneInstanceAdd(
    const RBSceneId SceneId,
    const RBGeomId GeomId,
    const struct SRBFloat44 Matrix,
    void *ClientData,
    const RBLayerId Layers,
    RBInstanceId * InstanceId);
```

**Description**
Register an instance of a registered geometry in a RayBooster 4 scene.

**Parameters**

*SceneId*
RBSceneId receiving the instance.

*GeomId*
Registered geometry to instantiate.

*Matrix*
Transformation matrix to apply to the newly instantiated geometry.

*ClientData*
User data pointer to associate to the newly instantiated geometry.

*Layers*
Composition of layers where the newly instantiated geometry is visible.

*InstanceId*
Pointer on a RBInstanceId to fill with the newly created unique instance identifier.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Instance was successfully added.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or GeomId is not valid or InstanceId is NULL.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>No more instance unique identifier available.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 17: List of all SceneInstanceAdd result code values, and their description.
Example

```c
struct SRBFloat44 instanceMatrix = RBFloat44Identity;
void* clientData = (void*)myData;
RBLayerId layersId = RBLayersUnion(RBLayer0, RBLayer3);
RBInstanceId instanceID = RBInvalidId;
enum ERBResult RBResult = driver.SceneInstanceAdd
    ( sceneId
    , geomId
    , instanceMatrix
    , clientData
    , layersId
    , &instanceID );
```
enum ERBResult SceneInstanceRemove(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId );

Description
Remove a geometry instance from a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the instance.

InstanceId
RBInstanceId to remove.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Instance was successfully removed.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 18: List of all SceneInstanceRemove result code values and their description.

Example

enum ERBResult RBResult =
    driver.SceneInstanceRemove(sceneId, instanceID);
enum ERBResult SceneInstanceMatrixSet(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId,
    const struct SRBFloat44 Matrix);

**Description**
Set the transformation matrix of a registered geometry instance in a RayBooster 4 scene.

**Parameters**
*SceneId*
RBSceneId containing the instance.

*InstanceId*
RBInstanceId identifying the instance.

*Matrix*
The transformation matrix to set.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Transformation matrix was successfully set.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 19: List of all SceneInstanceMatrixSet result code values, and their description.

**Example**

```c
struct SRBFloat44 instanceMatrix = RBFloat44Identity;
enum ERBResult RBRresult = driver.SceneInstanceMatrixSet
    (sceneId, instanceID, instanceMatrix);
```
enum ERBResult SceneInstanceMatrixGet(
   const RBSceneId SceneId,
   const RBInstanceId InstanceId,
   struct SRBFloat44* Matrix);

Description
Retrieve the transformation matrix of a registered geometry instance in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the instance.

InstanceId
RBInstanceId identifying instance.

Matrix
Pointer to a transformation matrix to fill.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Transformation matrix was successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid or Matrix is NULL.</td>
</tr>
</tbody>
</table>

Table 20: List of all SceneInstanceMatrixGet result code values, and their description.

Example

struct SRBFloat44 instanceMatrix;
enum ERBResult RBRresult = driver.SceneInstanceMatrixGet( sceneId, instanceID, &instanceMatrix );
```c
enum ERBResult SceneInstanceLayersSet(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId,
    const RBLayerId LayerIds);
```

**Description**
Set a set of active layers to a registered geometry instance in a RayBooster 4 scene.

**Parameters**
- **SceneId**
  RBSceneId containing the instance.

- **InstanceId**
  RBInstanceId identifying the instance.

- **LayerIds**
  Active layers to affect.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Active layers were successfully set.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 21: List of all `SceneInstanceLayersSet` result code values, and their description.

**Example**

```c
RBLayerId layersId = RBAAllLayers;
enum ERBResult RBRresult =
    driver.SceneInstanceLayersSet(SceneId, instanceID, layersId);
```
enum ERBResult SceneInstanceLayersGet(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId,
    RBLayerId* LayerIds );

Description
Retrieve the active layers for a registered geometry instance in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the instance.

InstanceId
RBInstanceId identifying the instance.

LayerIds
Pointer to a RBLayerId to fill.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Active layers were successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid or LayerIds is NULL.</td>
</tr>
</tbody>
</table>

Table 22: List of all SceneInstanceLayersGet result code values, and their description.

Example

RBLayerId layersId;
enum ERBResult RBResult = driver.SceneInstanceLayersGet(
    sceneId, instanceID, &layersId );
enum ERBRResult SceneInstanceGeometryGet(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId,
    RBGeomId* GeomId);

Description
Retrieve the unique geometry identifier of a registered geometry instance in a RayBooster 4 scene.

Parameters

SceneId
RBSceneId containing the instance.

InstanceId
RBInstanceId identifying the instance.

GeomId
Pointer to the RBGeomId to fill.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBRResult_Success</td>
<td>Geometry identifier was successfully retrieved.</td>
</tr>
<tr>
<td>ERBRResult_BadArgument</td>
<td>SceneId or InstanceId is not valid or GeomId is NULL.</td>
</tr>
</tbody>
</table>

Table 23: List of all SceneInstanceGeometryGet result code values, and their description.

Example

RBGeomId geomId;
enum ERBRResult RBRresult = driver.SceneInstanceGeometryGet
    (sceneId, instanceID, &geomId);
enum ERBResult SceneInstanceClientDataSet(
    const RBSceneId SceneId,
    const RBInstanceId InstanceId,
    void * ClientData );

Description
Affect a user data pointer to a registered geometry instance in a RayBooster 4 scene.

Parameters
SceneId
RBSceneId containing the geometry.

InstanceId
RBInstanceId identifying the geometry.

ClientData
User data pointer to set.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>User data pointer was successfully affected.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is locked.</td>
</tr>
</tbody>
</table>

Table 24: List of all SceneInstanceClientDataSet result code values and their description.

Example

```c
void * clientData = (void *)&myData;
eenum ERBResult RBRresult =
    driver.SceneInstanceClientDataSet(sceneId, instanceId, clientData);
```
enum ERBResult SceneInstanceClientDataGet(
   const RBSceneId SceneId,
   const RBInstanceId InstanceId,
   void ** ClientData );

**Description**
Retrieve user data pointer associated to a registered geometry instance in a RayBooster 4 scene.

**Parameters**
*SceneId*
RBSceneId containing the geometry.

*InstanceId*
RBInstanceId identifying geometry instance associated to required data pointer.

*ClientData*
Pointer to fill with the user data pointer.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>User data pointer was successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId or InstanceId is not valid or ClientData is NULL.</td>
</tr>
</tbody>
</table>

Table 25: List of all SceneInstanceClientDataGet result code values and their description.

**Example**

```c
void * clientData = NULL;
enum ERBResult RBResult = driver.SceneInstanceClientDataGet
   (sceneId , instanceId , &clientData );
```
3.3.5 Ray Tracing

```c
enum ERBResult SceneBuild
    ( const RBSSceneId SceneId );
```

**Description**
Build a RayBooster 4 scene in order to start a ray tracing session.

**Parameters**
- `SceneId`
RBSSceneId identifying the scene to build.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene was successfully built.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td><code>SceneId</code> is not valid.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>Scene is irretrievably damaged.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is not locked.</td>
</tr>
</tbody>
</table>

Table 26: List of all `SceneBuild` result code values and their description.

**Example**

```c
enum ERBResult RBRresult = driver.SceneBuild(sceneId);
```
**enum** ERBResult SceneBoxGet(
    const RBSceneId SceneId,
    struct SRBBox *Box );

**Description**
Retrieve the RayBooster 4 scene AABB. This box can be slightly larger than the declared geometry, in order to ensure that even with the numerical imprecision, all RayBooster 4 impacts are contained in this bounding box.

**Parameters**

*SceneId*
RBSceneId identifying the scene to build.

*Box*
Pointer to the SRBBox to fill.

**Errors**

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene bounding box was successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid or Box is NULL.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is not locked and/or not built.</td>
</tr>
</tbody>
</table>

Table 27: List of all SceneBoxGet result code values and their description.

**Example**

```c
struct SRBBox sceneBox;
enum ERBResult RBResult = driver.SceneBoxGet(SceneId, &sceneBox);
```
enum ERBRResult  SceneLayersBoxCompute(
    const RBSceneId  SceneId,
    const RBLayerId  Layers,
    struct SRBBox *Box );

Description
Compute the RayBooster 4 scene AABB of a set of specified layers. This box can be slightly larger than the declared geometry, in order to ensure that even with the numerical imprecision, all RayBooster 4 impacts are contained in this bounding box.

Parameters
(SceneId
RBSceneId identifying the scene to build.

Layers
Active RBLayerId that will be used for the AABB computation.

Box
Pointer to the SRBBox to fill.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Scene bounding box was successfully retrieved.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid or Box is NULL.</td>
</tr>
<tr>
<td>ERBResult_MemoryError</td>
<td>Scene is irretrievably damaged.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is not locked and/or not built.</td>
</tr>
</tbody>
</table>

Table 28: List of all SceneLayersBoxCompute result code values and their description.

Example

```
struct SRBBox sceneLayersBox;
RBLayerId activeLayers = RBLayersRemove(RBAllLayers, RBLayer0);
enum ERBRResult RBResult = driver.SceneLayersBoxCompute
    (sceneId, activeLayers, &sceneLayersBox);
```
enum ERBResult SceneTraceRay (  
    const RBSceneId SceneId,  
    const struct SRBRay Ray,  
    struct SRBHit *Hit,  
    const RBLayerId Layers );

Description
Intersect a single ray with a specified scene and a set of active layers.

Parameters
SceneId
RBSceneId identifying the scene to build.

Ray
Single ray description.

Hit
Pointer to the SRBHit to fill.

Layers
Active layers in the scene.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Ray was successfully intersected.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid or Hit is NULL.</td>
</tr>
<tr>
<td>ERBResult_IOError</td>
<td>Scene is not locked and/or not built.</td>
</tr>
</tbody>
</table>

Table 29: List of all SceneTraceRay result code values and their description.

Example

struct SRBRow ray(rayOrigin, rayDirection);
struct SRBHit hit;
RBLayerId layersId = RBAIILayers;
enum ERBResult RBResult =
    driver.SceneTraceRay(SceneId, ray, &hit, layersId);
enum ERBResult SceneTraceRays(
    const RBSceneId SceneId,
    const struct SRBRay* Rays,
    struct SRBHit* Hits,
    const RBNat32 NBRays,
    RBNat32 *NBHits,
    const RBLayerId Layers );

Description
Intersect a list of rays with a specified scene and a set of active layers.

Parameters
SceneId
RBSceneId identifying the scene to build.

Rays
Pointer to a SRBRay list.

Hits
Pointer to the SRBHit list to fill.

NBRays
Number of rays to use from the Rays list.

NBHits
Pointer to the number of intersection to fill.

Layers
Active layers in the scene.

Errors

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERBResult_Success</td>
<td>Rays were successfully intersected.</td>
</tr>
<tr>
<td>ERBResult_BadArgument</td>
<td>SceneId is not valid or Rays/ Hits/ NBHits is NULL.</td>
</tr>
<tr>
<td>ERBResult.IOError</td>
<td>Scene is not locked and/or not built.</td>
</tr>
</tbody>
</table>

Table 30: List of all SceneTraceRays result code values and their description.
Example

```c
const struct SRBRays rays[2] =
{ {raysOrigins[0], raysDirections[0]}
, {raysOrigins[1], raysDirections[1]} };  
struct SRBHits hits[2];  
const RBNat32 nbRays = 2;
RBNat32 nbIntersect = 0;
RBLayerId layersId = RBAllLayers;
enum ERBRResult RBResult = driver.SceneTraceRays
    (sceneId, rays, hits, nbRays, &nbIntersect, layersId);
```
enum ERBResult SceneTraceMany(
    const RBSceneId SceneId,
    const struct SRBFloat3 *RaysOrigin,
    const struct SRBFloat3 *RaysDirection,
    const struct SRBFloat2 *RaysRange,
    const struct SRBPrimitiveId *RaysInvalidPrimitiveId,
    struct SRBHit *Hits,
    const RBNat32 *NBHits,
    const RBTraceOptions TraceOptions,
    const RBLayerId Layers
);

Description
Intersect a list of rays defined with primitive types with a specified scene and a set of active layers.

Parameters
SceneId
RBSceneId identifying the scene to build.

RaysOrigin
Pointer to rays origins.

RaysDirection
Pointer to rays directions.

RaysRange
Pointer to rays validity ranges.

RaysInvalidPrimitiveId
Pointer to rays invalid primitive identifiers, may be NULL to indicate that no rays specify an invalid primitive identifier.

Hits
Pointer to the SRBHit list to fill.

NBHits
Number of rays to intersect in the list.
Pointer to the number of intersection found to fill.

*TraceOptions*
Trace options for the current function call, composition of available `RBTraceOptions`, described in the table 31.

<table>
<thead>
<tr>
<th>Trace option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RBTraceOptionNone</code></td>
<td>No particular option, rays don’t share any data.</td>
</tr>
<tr>
<td><code>RBTraceOptionSingleOrigin</code></td>
<td>Rays share the first origin.</td>
</tr>
<tr>
<td><code>RBTraceOptionSingleDirection</code></td>
<td>Rays share the first directions.</td>
</tr>
<tr>
<td><code>RBTraceOptionSingleRange</code></td>
<td>Rays share the first validity range.</td>
</tr>
<tr>
<td><code>RBTraceOptionSingleInvalidPrimitiveId</code></td>
<td>Rays share the first invalid primitive identifier.</td>
</tr>
</tbody>
</table>

Table 31: List of all composable trace options.

*Layers*
Active layers in the scene.

*Errors*
Result code value and their description.

<table>
<thead>
<tr>
<th>Result code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ERBResult_Success</code></td>
<td>Rays were successfully intersected.</td>
</tr>
<tr>
<td><code>ERBResult_BadArgument</code></td>
<td>Scene Id is not valid or one of the input pointers is NULL.</td>
</tr>
<tr>
<td><code>ERBResult_IOError</code></td>
<td>Scene is not locked and/or not built.</td>
</tr>
</tbody>
</table>

Table 32: List of all `SceneTraceMany` result code values and their description.

*Example*

```c
struct SRBHit hits[2];
const RBNat32 nbRays = 2;
RBNat32 nbIntersect = 0;
const RBTraceOptions options =
  RBTraceOptionSingleOrigin |  
  RBTraceOptionSingleRange | 
  RBTraceOptionSingleInvalidPrimitiveId;
RBLayerId layersId = RBAAllLayers;
enum ERBResult RBRresult = driver -> SceneTraceMany
  (sceneId, &raysOrigin, raysDirections, &range, NULL, hits
   ,nbRays, &nbIntersect, options, layersId);
```

37
4 Threading

The threading policies depend on the chosen RayBooster 4 driver. For the currently released CPU version, thread handling and scheduling are external to the driver. It means that critical functions are thread-safe, and no thread is created inside the driver. There is only one restriction: scene modifications are not thread-safe for a given scene.

Users can safely:

- Handle different drivers in different threads.
- Handle different scenes, inside the same driver or not, in different threads.

Inside a scene, RayBooster 4 can handle efficiently multiple threads for the build function and the trace function:

- Each thread entering the build method will be exploited, and will exit the function only when the build is effective.
- Each thread can safely call the trace function without considering other threads.