Geometry Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

About This Document

The purpose of this document is to provide an overview of the geometry library and describe its use from a developer's point of view.

Typographic Conventions

The typographic conventions used in this guide are explained in this section.

Hyperlinks

Hyperlinks are used to help you to navigate around the document. To return to where you clicked a hyperlink, select **View > Toolbars > More Tools** from the Adobe Reader main menu, and then enable the **Previous View** and **Next View** buttons.

Hints

A GUI shortcut or other useful tip for gaining maximum use from the software is presented as a "hint" surrounded by a box. For example:

> **Hint:** Example hint.

Notes

Additional advice or related information is presented as a "note" surrounded by a box. For example:

> **Note:** Example note.

Text

File names, source code, and command-line text are formatted in a fixed-width font. For example:

```
sce_geometry.h.
```

Errata

Any updates or amendments to this guide can be found in the release notes that accompany the release.

1 Library Overview

Characteristics

The geometry library provides classes for basic geometric primitives and associated functions that are useful for geometric calculations; for example, in collision detection, ray-tracing, and so forth.

Files

The following file is required to use the geometry library:

Table 1 Prerequisite File

File Name	Description
`target/include_common/sce_geometry.h`	Header file for API

Note: All functions are inlined; therefore, there are no `.a` or source files.

The user should not directly include the `closest.h`, `closest_implementation.h`, `intersect.h`, `intersect_implementation.h`, `types.h`, or `types_implementation.h` header files; they are included from the `sce_geometry.h` header file.

Sample Programs

The following sample programs show the basic usage of the geometry library:

```
target/samples/sample_code/system/api_sce_geometry/basic
target/samples/sample_code/system/api_sce_geometry/collision
```

2 How to Use the Library

Vector Representation

The geometry library uses matrix, 3D, and 4D vector types from the Vector Math Library. The current library release contains only geometric types built on AoS (Array of Structures) types but will work with the SIMD or scalar variants.

By default, the SIMD variant of Vector Math is included by `sce_geometry.h`. To use the scalar variant, add the following define before including `sce_geometry.h`:

```
#define SCE_GEOMETRY_USE_SCALAR_MATH
#include <sce_geometry.h>
```

Debugging and Asserts

By default, the geometry library uses the debug library (`libdbg`) to provide assert debugging macros. To enable the debug library assert macros, you must add the define `SCE_DBG_ASSERTS_ENABLED` before including `sce_geometry.h`:

```
#define SCE_DBG_ASSERTS_ENABLED 1
#include <sce_geometry.h>
```

Alternatively, you can customize the geometry library's assert functionality by adding a custom assert macro before including `sce_geometry.h`:

```
#define SCE_GEOMETRY_ASSERT(condition)    SOME_CUSTOM_ASSERT(condition)
#include <sce_geometry.h>
```

Note that when the geometry library's assert functionality is disabled but your code nevertheless uses such an assert, the resulting functionality is undefined if this assert would have failed.

Floating-Point Validity

Input Data

The geometry library is not designed to manage floating-point input data with invalid values such as `NaN` or `Inf`. The results of using such input values are undefined.

Exceptions

The geometry library is not compatible with floating-point exception trapping. Many of the operations are optimized for speed and do not always sanitize working data before division, normalization, and so forth.

API

Classes

The classes available within the geometry library are listed in Table 2:

Table 2 Geometry Library Classes

Class	Description
`Line`	A 3D line represented by a point on the line and a normalized direction along it.
`Ray`	A 3D ray represented by the starting (origin) point of the ray and a normalized, positive direction along it.
`Segment`	A 3D line segment represented by the starting (origin) point of the segment and a span to the end point.
`Plane`	A 3D plane represented by the plane normal vector and the plane dot-product scalar value.
`Sphere`	A 3D sphere represented by the center (origin) point and the radius.
`Capsule`	A 3D swept-sphere capsule represented by two points and the radius.
`Bounds`	A 3D axis aligned bounding volume represented by the minimum and maximum coordinate values along each axis.
`Aabb` (axis-aligned bounding box)	A 3D axis-aligned bounding box represented by a center (origin) point and the half-widths.
`Obb` (oriented bounding box)	A 3D oriented bounding box represented by a center (origin) point/orientation as a transform and the half-widths.
`Frustum`	A 3D frustum represented by six bounding planes.

Note: The specifications of the current internal representations of these classes are provided for informational purposes only. The internal implementations may change in the future; therefore, the user should not rely on these specifications.

Functions

The library provides a set of functions to calculate:

- Whether a primitive contains a point
- Whether two geometric primitives are intersecting (for various combinations of primitives)
- The intersection point of two geometric primitives (for various combinations of primitives)