Physics Effects Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

Sort Data

Structures

PfxSortData16, PfxSortData32

Sort data structure

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
struct SCE_PFX_ALIGNED(16) PfxSortData16 {
        union {
              PfxUInt8 i8data[16];
              PfxUInt16 i16data[8];
              PfxUInt32 i32data[4];
        };
};
struct SCE_PFX_ALIGNED(16) PfxSortData32 {
        union {
              PfxUInt8 i8data[2][16];
              PfxUInt16 i16data[2][8];
              PfxUInt32 i32data[2][4];
        };
};
```

Description

This is a sort data structure used to perform sorting with the `pfxParallelSort()` sort function. `PfxSortData16` has either 8-bit value x 16, 16-bit value x 8, or 32-bit value x 4 slots. `PfxSortData32` has either 8-bit value x 32, 16-bit value x 16, or 32-bit x 8 slots. The last 32 bits are always used to save the key value for sorting.

16-byte alignment is required.

Method List

Method	Description
`get8`	Gets an 8-bit value from the specified slot.
`set8`	Sets an 8-bit value into the specified slot.
`get16`	Gets a 16-bit value from the specified slot.
`set16`	Sets a 16-bit value into the specified slot.
`get32`	Gets a 32-bit value from the specified slot.
`set32`	Sets a 32-bit value into the specified slot.

List of Functions

Function	Description
`pfxGetKey()`	Get sort key value
`pfxSetKey()`	Set sort key value

Public Methods

get8, get16, get32

Get value of specified slot

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
struct SCE_PFX_ALIGNED(16) PfxSortData16 {
        PfxUInt8 get8(int slot) const;
        PfxUInt16 get16(int slot) const;
        PfxUInt32 get32(int slot) const;
);
struct SCE_PFX_ALIGNED(16) PfxSortData32 {
        PfxUInt8 get8(int slot) const;
        PfxUInt16 get16(int slot) const;
        PfxUInt32 get32(int slot) const;
);
```

Arguments

slot	Slot whose value is to be retrieved

Return Values

Returns the value of the specified slot.

Description

This public method gets the value of the specified slot.

The supported *slot* ranges are listed below.

Sort Data	Method	*slot* Range
PfxSortData16	get8()	0 to 15
	get16()	0 to 7
	get32()	0 to 3
PfxSortData32	get8()	0 to 31
	get16()	0 to 15
	get32()	0 to 7

set8, set16, set32

Set value into specified slot

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
struct SCE_PFX_ALIGNED(16) PfxSortData16 {
        void set8(int slot,PfxUInt8 data);
        void set16(int slot,PfxUInt16 data);
        void set32(int slot,PfxUInt32 data);
);
struct SCE_PFX_ALIGNED(16) PfxSortData32 {
        void set8(int slot,PfxUInt8 data);
        void set16(int slot,PfxUInt16 data);
        void set32(int slot,PfxUInt32 data);
);
```

Arguments

slot	Slot into which value is to be set
data	Value to be set

Return Values

None

Description

This public method sets the value into the specified slot.

The supported *slot* ranges are listed below.

Sort Data	Method	*slot* Range
PfxSortData16	set8()	0 to 15
	set16()	0 to 7
	set32()	0 to 3
PfxSortData32	set8()	0 to 31
	set16()	0 to 15
	set32()	0 to 7

Functions

pfxGetKey

Get value of sort key

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
SCE_PFX_FORCE_INLINE PfxUInt32 pfxGetKey(
        const PfxSortData16 &sortData
);
SCE_PFX_FORCE_INLINE PfxUInt32 pfxGetKey(
        const PfxSortData32 &sortData
);
```

Arguments

sortData	Sort data

Return Values

Returns the value of the sort key.

Description

This function gets the value of the sort key.

pfxSetKey

Set value of sort key

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
SCE_PFX_FORCE_INLINE void pfxSetKey(
        const PfxSortData16 &sortData,
        PfxUInt32 key
);
SCE_PFX_FORCE_INLINE void pfxSetKey(
        const PfxSortData32 &sortData,
        PfxUInt32 key
);
```

Arguments

sortData	Sort data
key	Value of sort key

Return Values

None

Description

This function sets the value of the sort key.

Broadphase Proxy

Structures

PfxBroadphaseProxy

Broadphase proxy structure

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_proxy.h>
typedef PfxSortData32 PfxBroadphaseProxy;
```

Description

This is the `PfxBroadphaseProxy` structure, which is the simple expression of the objects (rigid bodies) handled in the broad phase. It sets parameters from the rigid body data to the broadphase proxy using the `pfxUpdateBroadphaseProxy()` function. Updating before the broad phase is required if the position, shape, behavior, or contact filters of rigid bodies have changed.

16-byte alignment is required.

Broadphase Pair

Structures

PfxBroadphasePair

Broadphase pair structure

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
typedef PfxSortData16 PfxBroadphasePair;
```

Description

This is the `PfxBroadphasePair` structure, which expresses a pair of objects that are highly likely to collide detected from broad phase. A copy of the two broadphase proxy data forming the pair is kept and given as the input parameter for detection of the next collision. The index to the collision data must be set prior to the execution of collision detection.

16-byte alignment is required.

List of Functions

Function	Description
`pfxGetObjectIdA()`	Get index of object A
`pfxSetObjectIdA()`	Set index of object A
`pfxGetObjectIdB()`	Get index of object B
`pfxSetObjectIdB()`	Set index of object B
`pfxGetActive()`	Get active flag
`pfxSetActive()`	Set active flag
`pfxGetContactId()`	Get index to collision data
`pfxSetContactId()`	Set index to collision data

Functions

pfxGetObjectIdA, pfxGetObjectIdB

Get index of objects that form a pair

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE PfxUInt16 pfxGetObjectIdA(
        const PfxBroadphasePair &pair
);
SCE_PFX_FORCE_INLINE PfxUInt16 pfxGetObjectIdB(
        const PfxBroadphasePair &pair
);
```

Arguments

pair	Broadphase pair structure

Return Values

Returns the index of the objects.

Description

This function gets the index of the objects that form a pair.

pfxSetObjectIdA, pfxSetObjectIdB

Set index of objects forming a pair

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE void pfxSetObjectIdA(
        const PfxBroadphasePair &pair,
        PfxUInt16 i
);
SCE_PFX_FORCE_INLINE void pfxSetObjectIdB(
        const PfxBroadphasePair &pair,
        PfxUInt16 i
);
```

Arguments

pair	Broadphase pair structure
i	Index of objects

Return Values

None

Description

This function sets the index of the objects that form a pair.

pfxGetActive

Get active flag

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE PfxBool pfxGetActive(
        const PfxBroadphasePair &pair
);
```

Arguments

pair	Broadphase pair structure

Return Values

Returns the active flag.

Description

This function gets the active flag.

pfxSetActive

Set active flag

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE void pfxSetActive(
        const PfxBroadphasePair &pair,
        PfxBool b
);
```

Arguments

pair	Broadphase pair structure
b	true if active, false if inactive

Return Values

None

Description

This function sets the active flag. If the active flag is false, the constraint solver operation is canceled.

pfxGetContactId

Get index to collision data

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE PfxUInt32 pfxGetContactId(
        const PfxBroadphasePair &pair
);
```

Arguments

pair	Broadphase pair structure

Return Values

Returns the index to the collision data.

Description

This function gets the index to the collision data.

pfxSetContactId

Set index to collision data

Definition

```
#include <physics_effects/base_level/broadphase/pfx_broadphase_pairy.h>
SCE_PFX_FORCE_INLINE void pfxSetContactId(
        const PfxBroadphasePair &pair,
        PfxUInt32 i
);
```

Arguments

pair	Broadphase pair structure
i	Index to collision data

Return Values

None

Description

This function sets the index to the collision data.

Constraint Pair

Structures

PfxConstraintPair

Constraint pair structure

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
typedef PfxSortData16 PfxConstraintPair;
```

Description

This is the `PfxConstraintPair` structure, which expresses the target pair of the constraint solver operation. Since This is compatible with `PfxBroadphasePair`, `PfxBroadphasePair` detected from the broad phase can be handled as is as `PfxConstraintPair`. Only collision data `PfxContactManifold` or joint `PfxJoint` can be handled as constraints.

16-byte alignment is required.

List of Functions

Function	Description
`pfxGetObjectIdA()`	Get index of object A
`pfxSetObjectIdA()`	Set index of object A
`pfxGetObjectIdB()`	Get index of object B
`pfxSetObjectIdB()`	Set index of object B
`pfxGetActive()`	Get active flag
`pfxSetActive()`	Set active flag
`pfxGetConstraintId()`	Get index to constraint
`pfxSetConstraintId()`	Set index to constraint

Functions

pfxGetObjectIdA, pfxGetObjectIdB

Get index of objects that form a pair

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE PfxUInt16 pfxGetObjectIdA(
        const PfxConstraintPair &pair
);
SCE_PFX_FORCE_INLINE PfxUInt16 pfxGetObjectIdB(
        const PfxConstraintPair &pair
);
```

Arguments

pair	Constraint pair structure

Return Values

Returns the index of the objects.

Description

This function gets the index of the objects that form a pair.

pfxSetObjectIdA, pfxSetObjectIdB

Set index of objects that form a pair

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE void pfxSetObjectIdA(
        const PfxConstraintPair &pair,
        PfxUInt16 i
);
SCE_PFX_FORCE_INLINE void pfxSetObjectIdB(
        const PfxConstraintPair &pair,
        PfxUInt16 i
);
```

Arguments

pair	Constraint pair structure
i	Index of objects

Return Values

None

Description

This function sets the index of the objects that form a pair.

pfxGetActive

Get active flag

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE PfxBool pfxGetActive(
        const PfxConstraintPair &pair
);
```

Arguments

pair	Constraint pair structure

Return Values

Returns the active flag.

Description

This function gets the active flag.

pfxSetActive

Set the active flag

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE void pfxSetActive(
        const PfxConstraintPair &pair,
        PfxBool b
);
```

Arguments

pair	Constraint pair structure
b	true if active, false if inactive

Return Values

None

Description

This function sets the active flag. If the active flag is false, the constraint solver operation is canceled.

pfxGetConstraintId

Get index to constraint

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE PfxUInt32 pfxGetConstraintId(
        const PfxConstraintPair &pair
);
```

Arguments

pair	Constraint pair structure

Return Values

Returns the index to the constraint.

Description

This function gets the index to the constraint.

pfxSetConstraintId

Set index to constraint

Definition

```
#include <physics_effects/base_level/solver/pfx_constraint_pair.h>
SCE_PFX_FORCE_INLINE void pfxSetConstraintId(
        const PfxConstraintPair &pair,
        PfxUInt32 i
);
```

Arguments

pair	Constraint pair structure
i	Index to constraint

Return Values

None

Description

This function sets the index to the constraint.

Rigid Body State

Classes

PfxRigidState

State of rigid body

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {};
```

Description

This class represents the current state of the rigid body. It stores data such as position, velocity, and contact filter.

128-byte alignment is required.

Method List

Method	Description
`reset`	Resets the parameters to their initial values
`getRigidBodyId`	Gets the rigid body index
`setRigidBodyId`	Sets the rigid body index
`getContactFilterSelf`	Gets the self contact filter
`setContactFilterSelf`	Sets the self contact filter
`getContactFilterTarget`	Gets the target contact filter
`setContactFilterTarget`	Sets the target contact filter
`getMotionType`	Gets the motion type
`setMotionType`	Sets the motion type
`getLinearDamping`	Gets the damping value of linear motion
`setLinearDamping`	Sets the damping value of linear motion
`getAngularDamping`	Gets the damping value of angular motion
`setAngularDamping`	Sets the damping value of angular motion
`getPosition`	Gets the position
`setPosition`	Sets the position
`getOrientation`	Gets the orientation
`setOrientation`	Sets the orientation
`getLinearVelocity`	Gets the linear velocity
`setLinearVelocity`	Sets the linear velocity
`getAngularVelocity`	Gets the angular velocity.
`setAngularVelocity`	Sets the angular velocity.
`movePosition`	Sets the linear velocity for moving to the specified position after the specified time
`moveOrientation`	Sets the angular velocity for rotating to the specified orientation after the specified time
`isAsleep`	Returns whether the rigid body is in the sleep mode
`isAwake`	Returns whether the rigid body is in the active mode
`wakeup`	Wakes up the rigid body
`sleep`	Puts the rigid body to sleep
`getUseSleep`	Gets the sleep flag
`setUseSleep`	Sets the sleep flag
`incrementSleepCount`	Increments the sleep count by 1

Method	Description
`resetSleepCount`	Resets the sleep count to 0
`getSleepCount`	Gets the sleep count
`getMaxLinearVelocity`	Gets the maximum linear velocity
`setMaxLinearVelocity`	Sets the maximum linear velocity
`getMaxAngularVelocity`	Gets the maximum angular velocity
`setMaxAngularVelocity`	Sets the maximum angular velocity
`getUserData`	Gets the user data
`setUserData`	Sets the user data

Public Methods

reset

Set parameters to their initial values

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        inline void reset();
};
```

Arguments

None

Return Values

None

Description

This public method initializes the rigid body parameters.

getRigidBodyId

Get rigid body index

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt16 getRigidBodyId() const;
};
```

Arguments

None

Return Values

Returns the index of the rigid body.

Description

This public method gets the index of the rigid body.

setRigidBodyId

Set rigid body index

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setRigidBodyId(PfxUInt16 i);
};
```

Arguments

i	Index of rigid body

Return Values

None

Description

This public method sets the index of the rigid body.

getContactFilterSelf

Get self contract filter

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt32 getContactFilterSelf() const;
};
```

Arguments

None

Return Values

Returns the self contact filter.

Description

This public method gets the self contact filter.

setContactFilterSelf

Set self contact filter

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setContactFilterSelf(PfxUInt32 filter);
};
```

Arguments

filter	Self contact filter

Return Values

None

Description

This public method sets the self contact filter.

getContactFilterTarget

Get target contact filter

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt32 getContactFilterTarget() const;
};
```

Arguments

None

Return Values

Returns the target contact filter.

Description

This public method gets the target contact filter.

setContactFilterTarget

Set target contact filter

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setContactFilterTarget(PfxUInt32 filter);
};
```

Arguments

filter	Target contact filter

Return Values

None

Description

This public method sets the target contact filter.

getMotionType

Get motion type

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        ePfxMotionType getMotionType() const;
};
```

Arguments

None

Return Values

Returns the motion type.

Description

This public method gets the motion type.

setMotionType

Set motion type

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setMotionType(ePfxMotionType t);
};
```

Arguments

t	Motion type

Return Values

None

Description

This public method sets the motion type.

If the range of the argument value is invalid, an assert is called.

getLinearDamping

Get damping value of linear motion

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxFloat getLinearDamping() const;
};
```

Arguments

None

Return Values

Returns the damping value of the linear motion.

Description

This public method gets the damping value of the linear motion.

setLinearDamping

Set damping value of linear motion

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setLinearDamping(PfxFloat damping);
};
```

Arguments

damping	Damping value of linear motion

Return Values

None

Description

This public method sets the damping value of the linear motion.

getAngularDamping

Get damping value of angular motion

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxFloat getAngularDamping() const;
};
```

Arguments

None

Return Values

Returns the damping value of the angular motion.

Description

This public method gets the damping value of the angular motion.

setAngularDamping

Set damping value of angular motion

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setAngularDamping(PfxFloat damping);
};
```

Arguments

damping	Damping value of angular motion

Return Values

None

Description

This public method sets the damping value of the angular motion.

getPosition

Get position

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxVector3 getPosition() const;
};
```

Arguments

None

Return Values

Returns the position.

Description

This public method gets the position.

setPosition

Set position

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setPosition(const PfxVector3 &pos);
};
```

Arguments

pos	Position

Return Values

None

Description

This public method sets the position.

getOrientation

Get orientation

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxQuat getOrientation() const;
};
```

Arguments

None

Return Values

Returns the orientation.

Description

This public method gets the orientation.

setOrientation

Set orientation

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setOrientation(const PfxQuat &rot);
};
```

Arguments

rot	Orientation

Return Values

None

Description

This public method sets the orientation.

getLinearVelocity

Get linear velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxVector3 getLinearVelocity() const;
};
```

Arguments

None

Return Values

Returns the linear velocity.

Description

This public method gets the linear velocity.

setLinearVelocity

Set linear velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setLinearVelocity(const PfxVector3 &vel);
};
```

Arguments

vel	Linear velocity

Return Values

None

Description

This public method sets the linear velocity.

getAngularVelocity

Get angular velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxVector3 getAngularVelocity() const;
};
```

Arguments

None

Return Values

Returns the angular velocity.

Description

This public method gets the angular velocity.

setAngularVelocity

Set angular velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setAngularVelocity(const PfxVector3 &vel);
};
```

Arguments

vel	Angular velocity

Return Values

None

Description

This public method sets the angular velocity.

movePosition

Set liner velocity for moving to specified position after specified time

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        inline void movePosition(const PfxVector3 &pos,PfxFloat timeStep);
};
```

Arguments

pos	Position after move
timeStep	Time step (second)

Return Values

None

Description

This public method sets the linear velocity for moving to the specified position after the specified time.

moveOrientation

Set angular velocity for rotating to specified orientation after specified time

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        inline void moveOrientation(const PfxQuat &rot,PfxFloat timeStep);
};
```

Arguments

rot	Orientation after rotation
timeStep	Time step (second)

Return Values

None

Description

This public method sets the angular velocity for rotating to the specified orientation after the specified time.

isAsleep

Returns whether the rigid body is in the sleep mode

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxBool isAsleep() const;
};
```

Arguments

None

Return Values

Returns true when the rigid body state is sleep.

Description

This public method returns whether the rigid body state is sleep.

isAwake

Return whether rigid body is in active mode

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxBool isAwake() const;
};
```

Arguments

None

Return Values

Returns true when the rigid body is active.

Description

This public method returns whether the rigid body is in the active mode.

wakeup

Wake up rigid body

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void wakeup();
};
```

Arguments

None

Return Values

None

Description

This public method releases the rigid body from the sleep mode and sets it to the active mode.

sleep

Put rigid body to sleep

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void sleep();
};
```

Arguments

None

Return Values

None

Description

This public method forcibly transitions the rigid body state to sleep.

Rigid bodies whose motion type is either `kPfxMotionTypeFixed` or `kPfxMotionTypeTrigger` cannot be set in the sleep state.

getUseSleep

Get sleep flag

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt8 getUseSleep() const;
};
```

Arguments

None

Return Values

Returns the sleep flag.

Description

Returns 1 when sleep is enabled, and 0 when sleep is disabled.

setUseSleep

Set sleep flag

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setUseSleep(PfxUInt8 b);
};
```

Arguments

b	Sleep enable (1) / disable (0)

Return Values

None

Description

This public method specifies whether to sleep for each rigid body state. To enable sleep, set 1, and to disable sleep, set 0.

incrementSleepCount

Increment sleep count by 1

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void incrementSleepCount();
};
```

Arguments

None

Return Values

None

Description

This public method increments the sleep count by 1.

resetSleepCount

Reset sleep count

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void resetSleepCount();
};
```

Arguments

None

Return Values

None

Description

This public method sets the sleep count to 0.

getSleepCount

Get sleep count

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt16 getSleepCount() const;
};
```

Arguments

None

Return Values

Returns the sleep count.

Description

This public method gets the sleep count.

getMaxLinearVelocity

Get maximum linear velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxFloat getMaxLinearVelocity() const;
};
```

Arguments

None

Return Values

Returns the maximum linear velocity.

Description

This public method gets the maximum linear velocity.

setMaxLinearVelocity

Set maximum linear velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setMaxLinearVelocity(PfxFloat maxVelocity);
};
```

Arguments

maxVelocity	Maximum linear velocity

Return Values

Sets the maximum linear velocity.

Description

When the velocity of a rigid body is updated with `pfxApplyExternalForce()`, if the linear velocity of the rigid body exceeds the maximum linear velocity, it will be reset to the maximum value.

getMaxAngularVelocity

Get the maximum angular velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxFloat getMaxAngularVelocity() const;
};
```

Arguments

None

Return Values

Returns the maximum angular velocity.

Description

This public method gets the maximum angular velocity.

setMaxAngularVelocity

Set maximum angular velocity

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setMaxAngularVelocity(PfxFloat maxVelocity);
};
```

Arguments

maxVelocity	Maximum angular velocity

Return Values

Sets the maximum angular velocity.

Description

When the velocity of a rigid body is updated with `pfxApplyExternalForce()`, if the angular velocity of the rigid body exceeds the maximum angular velocity, it will be reset to the maximum value.

getUserData

Get user data

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void *getUserData() const;
};
```

Arguments

None

Return Values

Returns the user data.

Description

This public method gets the user data.

setUserData

Set user data

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setUserData(void *d);
};
```

Arguments

d	User data

Return Values

Sets the user data.

Description

This public method sets the user data as a `void` type pointer.

getUserParam

Get user parameters

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        PfxUInt32 getUserParam(int i) const;
};
```

Arguments

i	Slot index

Return Values

Returns user parameters.

Description

This public method obtains user parameters. Four 32-bit integer values can be stored as user parameters. Specify 0, 1, 2, and/or 3 to the slot index and obtain the corresponding parameters.

setUserParam

Set user parameters

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_state.h>
class SCE_PFX_ALIGNED(128) PfxRigidState {
        void setUserParam(int i, PfxUInt32 param);
};
```

Arguments

i	Slot index
param	User parameter

Return Values

Sets user parameters.

Description

This public method sets the `param` parameters corresponding to the 0, 1, 2, and/or 3 slots specified to the slot index.

Rigid Body Attributes

Classes

PfxRigidBody

Rigid body attributes

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {};
```

Description

This is a class that expresses the physical attributes of a rigid body, such as mass and friction. The values of the parameters cannot be changed during simulation.

128-byte alignment is required.

Method List

Method	Description
`reset`	Resets the parameters to their initial values
`getMass`	Gets the mass
`setMass`	Sets the mass
`getInertia`	Gets the inertia tensor
`setInertia`	Sets the inertia tensor
`getRestitution`	Gets the restitution coefficient
`setRestitution`	Sets the restitution coefficient
`getFriction`	Gets the friction coefficient
`setFriction`	Sets the friction coefficient

Public Methods

reset

Reset parameters to their initial values

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        inline void reset();
};
```

Arguments

None

Return Values

None

Description

This public method initializes the parameters of the rigid body attributes.

getMass

Get mass

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        PfxFloat getMass() const;
};
```

Arguments

None

Return Values

Returns the mass.

Description

This public method gets the mass.

setMass

Set mass

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        void setMass(PfxFloat mass);
};
```

Arguments

mass	Mass

Return Values

None

Description

This public method sets the mass.

getInertia

Get inertia tensor

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        const PfxMatrix3& getInertia() const;
};
```

Arguments

None

Return Values

Returns the inertia tensor.

Description

This public method gets the inertia tensor.

setInertia

Set inertia tensor

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        void setInertia(
              const PfxMatrix3 SCE_VECTORMATH_AOS_MATRIX_ARG inertia
        );
};
```

Arguments

inertia	Inertia tensor

Return Values

None

Description

This public method sets the inertia tensor.

getRestitution

Get restitution coefficient

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        PfxFloat getRestitution() const;
};
```

Arguments

None

Return Values

Returns the restitution coefficient.

Description

This public method gets the restitution coefficient.

setRestitution

Set restitution coefficient

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        void setRestitution(PfxFloat restitution);
};
```

Arguments

restitution	Restitution coefficient

Return Values

None

Description

This public method sets the restitution coefficient.

getFriction

Get friction

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        PfxFloat getFriction() const;
};
```

Arguments

None

Return Values

Returns the friction.

Description

This public method gets the friction.

setFriction

Set friction

Definition

```
#include <physics_effects/base_level/rigidbody/pfx_rigid_body.h>
class SCE_PFX_ALIGNED(128) PfxRigidBody {
        void setFriction(PfxFloat friction);
};
```

Arguments

friction	Friction

Return Values

None

Description

This public method sets the friction. No distinction is made between dynamic friction and static friction.

Rigid Body Shape

Classes

PfxCollidable

Rigid body shape

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {};
```

Description

This is a class that expresses the shape of a rigid body. `PfxCollidable` functions as a container that stores several `PfxShape` expressing a single shape.

128-byte alignment is required.

Method List

Method	Description
`reset`	Resets the parameters to their initial values
`finish`	Completes shape registration
`addShape`	Adds a shape
`getNumShapes`	Gets the number of shapes
`getMaxNumShapes`	Gets the maximum number of shapes that can be registered
`getShape`	Gets the shape
`getShapeId`	Gets the index of the shape stored in the array
`getHalf`	Gets the size of the bounding box
`getCenter`	Gets the center of the bounding box

Public Methods

reset

Reset parameters to their initial values

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline void reset();
};
```

Arguments

None

Return Values

None

Description

This public method initializes the rigid body shape parameters.

reset

Reset parameters to their initial values

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline void reset(
              PfxShape *base,
              PfxUInt16 *ids,
              int n=1
        );
};
```

Arguments

base	Address of shape array
ids	Address of index array
n	Number of shapes

Return Values

None

Description

When two or more shapes are included, prepare an external array for storing the shapes and call `reset()`.

finish

Complete shape registration

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        void finish();
};
```

Arguments

None

Return Values

None

Description

After adding a shape, be sure to call `finish()` to complete the registration.

addShape

Add a shape

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        void addShape(const PfxShape &shape);
};
```

Arguments

shape	Shape

Return Values

None

Description

This public method adds a shape. The contents of the shape are copied.

getNumShapes

Get number of shapes

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline PfxUInt32 getNumShapes() const;
};
```

Arguments

None

Return Values

None

Description

This public method gets the number of shapes.

getMaxNumShapes

Get maximum number of shapes that can be registered

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline PfxUInt32 getMaxNumShapes() const;
};
```

Arguments

None

Return Values

None

Description

This public method gets the maximum number of shapes that can be registered.

getShape

Get shape

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline const PfxShape& getShape(int i) const;
        inline PfxShape& getShape(int i);
};
```

Arguments

i	Index of shape

Return Values

Returns the shape.

Description

This public method gets the shape. If the index exceeds the range, assert is called.

getShapeId

Get index of shape stored in array

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline PfxUInt16 getShapeId(int i) const;
};
```

Arguments

i	Index of shape

Return Values

Returns the index of the shape stored in the array.

Description

In the case of complex shapes, shape No. 0 and subsequent shapes are stored in an external shape array. This method is used for getting the index for the external shape array of the specified shape. If the argument index exceeds the range, assert is called.

getHalf

Get size of bounding box

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline PfxVector3 getHalf() const;
};
```

Arguments

None

Return Values

Returns the size of the bounding box.

Description

This public method gets the size of the bounding box.

getCenter

Get center of bounding box

Definition

```
#include <physics_effects/base_level/collision/pfx_collidable.h>
class SCE_PFX_ALIGNED(128) PfxCollidable {
        inline PfxVector3 getCenter() const;
};
```

Arguments

None

Return Values

Returns the center of the bounding box.

Description

This public method gets the center of the bounding box.

Shape

Classes

PfxShape

Shape class

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {};
```

Description

This class represents a single shape. Register it to `PfxCollidable` after setting the actual shape. 16-byte alignment is required.

Method List

Method	Description
`reset`	Resets the parameters to their initial values
`getType`	Gets the type of shape
`getAabb`	Gets the bounding box
`getSphere`	Gets a sphere shape
`setSphere`	Sets a sphere shape
`getBox`	Gets a box shape
`setBox`	Sets a box shape
`getCapsule`	Gets a capsule shape
`setCapsule`	Sets a capsule shape
`getCylinder`	Gets a cylinder shape
`setCylinder`	Sets a cylinder shape
`getConvexMesh`	Gets a convex mesh
`setConvexMesh`	Sets a convex mesh
`getLargeTriMesh`	Gets a large mesh
`setLargeTriMesh`	Sets a large mesh
`getOffsetTransform`	Gets an offset transform
`setOffsetTransform`	Sets an offset transform
`getOffsetPosition`	Gets the offset position
`setOffsetPosition`	Sets the offset position
`getOffsetOrientation`	Gets the offset orientation
`setOffsetOrientation`	Sets the offset orientation
`getContactFilterSelf`	Gets the self contact filter
`setContactFilterSelf`	Sets the self contact filter
`getContactFilterTarget`	Gets the target contact filter
`setContactFilterTarget`	Sets the target contact filter

Public Methods

reset

Reset parameters to their initial values

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void reset();
};
```

Arguments

None

Return Values

None

Description

This public method initializes the shape parameters.

getType

Get type of shape

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxUInt8 getType() const;
};
```

Arguments

None

Return Values

Returns the following shape types.

Value	Shape Type
`kPfxShapeSphere`	Sphere
`kPfxShapeBox`	Box
`kPfxShapeCapsule`	Capsule
`kPfxShapeCylinder`	Cylinder
`kPfxShapeConvexMesh`	Convex mesh
`kPfxShapeLargeTriMesh`	Large mesh

Description

This public method gets the shape type.

getAabb

Get size of bounding box

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        void getAabb(PfxVector3 &aabbMin,PfxVector3 &aabbMax) const;
};
```

Arguments

aabbMin	Minimum coordinate value of bounding box
aabbMax	Maximum coordinate value of bounding box

Return Values

None

Description

This public method stores the size of the bounding box in *aabbMin*, *aabbMax* given as arguments.

getSphere

Get sphere

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxSphere getSphere() const;
};
```

Arguments

None

Return Values

Returns a sphere.

Description

This public method gets a sphere.

setSphere

Set sphere

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setSphere(PfxSphere sphere);
};
```

Arguments

sphere	Sphere

Return Values

None

Description

This public method sets a sphere.

getBox

Get box

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxBox getBox() const;
};
```

Arguments

None

Return Values

Returns a box.

Description

This public method gets a box.

setBox

Set box

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setBox(PfxBox SCE_VECTORMATH_AOS_VECTOR_ARG box);
};
```

Arguments

box	Box

Return Values

None

Description

This public method sets a box.

getCapsule

Get capsule

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxCapsule getCapsule() const;
};
```

Arguments

None

Return Values

Returns the capsule.

Description

This public method gets a capsule.

setCapsule

Set capsule

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setCapsule(PfxCapsule capsule);
};
```

Arguments

capsule	Capsule

Return Values

None

Description

This public method sets a capsule.

getCylinder

Get cylinder

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxCylinder getCylinder() const;
};
```

Arguments

None

Return Values

Returns the cylinder.

Description

This public method gets a cylinder.

setCylinder

Set cylinder

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setCylinder(PfxCylinder cylinder);
};
```

Arguments

cylinder	Cylinder

Return Values

None

Description

This public method sets a cylinder.

getConvexMesh

Get convex mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline const PfxConvexMesh* getConvexMesh() const;
};
```

Arguments

None

Return Values

Returns the pointer to the convex mesh.

Description

This public method gets the pointer to the convex mesh.

setConvexMesh

Set convex mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setConvexMesh(const PfxConvexMesh *convexMesh);
};
```

Arguments

convexMesh	Pointer to convex mesh

Return Values

None

Description

This public method sets a convex mesh.

getLargeTriMesh

Get large mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline const PfxLargeTriMesh* getLargeTriMesh() const;
};
```

Arguments

None

Return Values

Returns the pointer to the large mesh.

Description

This public method gets the pointer to the large mesh.

setLargeTriMesh

Set large mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setLargeTriMesh(const PfxLargeTriMesh *largeMesh);
};
```

Arguments

largeMesh	Pointer to large mesh

Return Values

None

Description

This public method sets a large mesh.

getOffsetTransform

Get offset transform

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxTransform3 getOffsetTransform() const;
};
```

Arguments

None

Return Values

Returns an offset transform.

Description

This public method gets an offset transform. The offset is expressed in relation to the rigid body origin.

setOffsetTransform

Set offset transform

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setOffsetTransform(const PfxTransform3 &xfrm);
};
```

Arguments

xfrm	Offset transform

Return Values

None

Description

This public method sets an offset transform. The offset is expressed in relation to the rigid body origin.

getOffsetPosition

Get offset position

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxVector3 getOffsetPosition() const;
};
```

Arguments

None

Return Values

Returns the offset position.

Description

This public method gets the offset position. The offset is expressed in relation to the rigid body origin.

setOffsetPosition

Set offset position

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setOffsetPosition(
             const PfxVector3 SCE_VECTORMATH_AOS_VECTOR_ARG pos
        );
};
```

Arguments

pos	Offset position

Return Values

None

Description

This public method sets the offset position. The offset is expressed in relation to the rigid body origin.

getOffsetOrientation

Get offset orientation

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline PfxQuat getOffsetOrientation() const;
};
```

Arguments

None

Return Values

Returns the offset orientation.

Description

This public method gets the offset orientation. The offset is expressed in relation to the rigid body origin.

setOffsetOrientation

Set offset orientation

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        inline void setOffsetOrientation(
             const PfxQuat SCE_VECTORMATH_AOS_VECTOR_ARG rot
        );
};
```

Arguments

rot	Offset orientation

Return Values

None

Description

This public method sets the offset orientation. The offset is expressed in relation to the rigid body origin.

getContactFilterSelf

Get self contact filter

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        PfxUInt32 getContactFilterSelf() const;
};
```

Arguments

None

Return Values

Returns the self contact filter.

Description

This public method gets the self contact filter.

setContactFilterSelf

Set self contact filter

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        void setContactFilterSelf(PfxUInt32 filter);
};
```

Arguments

filter	Contact filter value

Return Values

None

Description

This public method sets the self contact filter.

getContactFilterTarget

Get target contact filter

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        PfxUInt32 getContactFilterTarget() const;
};
```

Arguments

None

Return Values

Returns the target contact filter.

Description

This public method gets the target contact filter.

setContactFilterTarget

Set target contact filter

Definition

```
#include <physics_effects/base_level/collision/pfx_shape.h>
class SCE_PFX_ALIGNED(16) PfxShape {
        void setContactFilterTarget(PfxUInt32 filter);
};
```

Arguments

filter	Contact filter value

Return Values

None

Description

This public method sets the target contact filter.

Sphere

Classes

PfxSphere

Sphere

Definition

```
#include <physics_effects/base_level/collision/pfx_sphere.h>
class PfxSphere {
        PfxFloat m_radius;
};
```

Members

m_radius	Radius of sphere

Description

This class expresses spheres.

Method List

Method	Description
PfxSphere	Creates a sphere with the specified radius

Constructors and Destructors

PfxSphere

Create sphere of specified radius

Definition

```
#include <physics_effects/base_level/collision/pfx_sphere.h>
class PfxSphere {
        PfxSphere(PfxFloat radius);
};
```

Arguments

radius	Radius of sphere

Return Values

None

Description

Creates a sphere of the specified radius.

Box

Classes

PfxBox

Box

Definition

```
#include <physics_effects/base_level/collision/pfx_box.h>
class PfxBox {
        PfxVector3 m_half;
};
```

Members

m_half	Half size of box

Description

This class expresses boxes.

Method List

Method	Description
PfxBox	Creates a box of the specified size

Constructors and Destructors

PfxBox

Create box of specified size

Definition

```
#include <physics_effects/base_level/collision/pfx_box.h>
class PfxBox {
        PfxBox(const PfxVector3 SCE_VECTORMATH_AOS_VECTOR_ARG half);
        PfxBox(PfxFloat hx, PfxFloat hy, PfxFloat hz);
};
```

Arguments

half	Half size of box
hx, *hy*, *hz*	Half size of box

Return Values

None

Description

Creates a box of the specified size.

Capsule

Classes

PfxCapsule

Capsule

Definition

```
#include <physics_effects/base_level/collision/pfx_capsule.h>
struct PfxCapsule {
        PfxFloat m_halfLen;
        PfxFloat m_radius;
};
```

Members

m_halfLen	Half length of capsule
m_radius	Radius of capsule

Description

This class expresses capsules. The axis of a capsule always follows the X axis. To change the direction, adjust the offset orientation given to `PfxShape`.

Method List

Method	Description
`PfxCapsule`	Creates a capsule of the specified radius and length

Constructors and Destructors

PfxCapsule

Create capsule of specified radius and length

Definition

```
#include <physics_effects/base_level/collision/pfx_capsule.h>
class PfxCapsule {
        PfxCapsule(PfxFloat halfLength, PfxFloat radius);
};
```

Arguments

halfLength	Half length of capsule
radius	Radius of capsule

Return Values

None

Description

Creates a capsule of the specified radius and length.

Cylinder

Classes

PfxCylinder

Cylinder

Definition

```
#include <physics_effects/base_level/collision/pfx_cylinder.h>
struct PfxCylinder {
        PfxFloat m_halfLen;
        PfxFloat m_radius;
};
```

Members

m_halfLen	Half length of cylinder
m_radius	Radius of cylinder

Description

This class expresses cylinders. The axis of a cylinder always follows the X axis. To change the direction, adjust the offset orientation given to `PfxShape`.

Method List

Method	Description
`PfxCylinder`	Creates a cylinder of the specified radius and length

Constructors and Destructors

PfxCylinder

Create cylinder of the specified radius and length

Definition

```
#include <physics_effects/base_level/collision/pfx_cylinder.h>
class PfxCylinder {
        PfxCylinder(PfxFloat halfLength, PfxFloat radius);
};
```

Arguments

halfLength	Half length of cylinder
radius	Radius of cylinder

Return Values

None

Description

Creates a cylinder of the specified radius and length.

Convex Mesh

Structures

PfxConvexMesh

Convex mesh structure

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxConvexMesh {
        PfxUInt8 m_numVerts;
        PfxUInt8 m_numIndices;
        PfxUInt16 *m_indices;
        PfxVector3 *m_verts;
        PfxVector3 m_half;
};
```

Members

m_numVerts	Number of vertices
m_numIndices	Number of triangle indexes
m_verts	Vertex array
m_indices	Index array
m_half	Size of bounding box

Description

This structure expresses convex meshes. To create it, use the `pfxCreateConvexMesh()` utility function.

16-byte alignment is required.

Method List

Method	Description
`PfxConvexMesh`	Initializes the convex mesh parameters
`updateAABB`	Calculates the size of the bounding box

Constructors and Destructors

PfxConvexMesh

Initialize convex mesh parameters

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxConvexMesh {
        PfxConvexMesh();
};
```

Arguments

None

Return Values

None

Description

Initializes the convex mesh parameters.

Public Methods

updateAABB

Calculate size of bounding box

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxConvexMesh {
        inline void updateAABB();
};
```

Arguments

None

Return Values

None

Description

This public method calculates the size of the bounding box and saves it to the *m_half* internal parameter.

Large Mesh

Structures

PfxQuantize, PfxQuantize2, PfxQuantize3

Quantization data structure

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxQuantize {
        PfxInt16 elem;
};
struct PfxQuantize2 {
        PfxInt16 elem[2];
};
struct PfxQuantize3 {
        PfxInt16 elem[3];
};
```

Members

elem	Quantization data

Description

This structure stores values that have been compressed through quantization.

Method List

Method	Description
PfxQuantize	Initializes the quantization data
PfxQuantize2	Initializes the quantization data
PfxQuantize3	Initializes the quantization data

Constructors and Destructors

PfxQuantize, PfxQuantize2, PfxQuantize3

Initialize quantization data

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxQuantize {
        PfxQuantize(PfxInt16 value);
};
struct PfxQuantize2 {
        PfxQuantize2(PfxInt16 value1, PfxInt16 value2);
};
struct PfxQuantize3 {
        PfxQuantize3(PfxInt16 value1, PfxInt16 value2, PfxInt16 value3);
};
```

Arguments

value	Quantization data
value1	Quantization data
value2	Quantization data
value3	Quantization data

Return Values

None

Description

This structure initializes the quantization data with the values assigned to the arguments.

Structures

PfxEdge

Edge structure included in island mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxEdge {
        PfxUInt8 m_vertId[2];
        PfxUInt8 m_angleType;
        PfxUInt8 m_tilt;
};
```

Members

m_vertId	Vertex index of end point
m_angleType	Edge type
m_tilt	Edge angle

Description

An edge represents the boundary of facets.

The types of edge are classified as follows according to the angle formed by two facets.

Value	Description
SCE_PFX_EDGE_FLAT	Flat edge
SCE_PFX_EDGE_CONVEX	Convex edge
SCE_PFX_EDGE_CONCAVE	Concave edge

Structures

PfxExpandedFacet, PfxQuantizedFacet

Facet structure that makes up island mesh of array type

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxExpandedFacet {
        PfxFloat3 m_normal;
        PfxFloat3 m_half;
        PfxFloat3 m_center;
        PfxFloat m_thickness;
        PfxUInt8 m_vertIds[3];
        PfxUInt8 m_edgeIds[3];
        PfxUInt32 m_userData;
};
struct PfxQuantizedFacet {
        PfxQuantize2 m_normal;
        PfxQuantize3 m_half;
        PfxQuantize3 m_center;
        PfxQuantize m_thickness;
        PfxUInt8 m_vertIds[3];
        PfxUInt8 m_edgeIds[3];
        PfxUInt32 m_userData;
};
```

Members

m_normal	Normal
m_half	Size
m_center	Center
m_thickness	Thickness of facet
m_vertIds	Vertex index
m_edgeIds	Edge index
m_userData	User data

Description

This is a facet structure that is created when the array type is specified during large mesh creation.

Structures

PfxExpandedFacetBvh, PfxQuantizedFacetBvh

Facet structure that makes up island mesh of BVH type

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct PfxExpandedFacetBvh {
        PfxFloat3 m_normal;
        PfxFloat m_thickness;
        PfxUInt8 m_vertIds[3];
        PfxUInt8 m_edgeIds[3];
        PfxUInt32 m_userData;
};
struct PfxQuantizedFacetBvh {
        PfxQuantize2 m_normal;
        PfxQuantize m_thickness;
        PfxUInt8 m_vertIds[3];
        PfxUInt8 m_edgeIds[3];
        PfxUInt32 m_userData;
};
```

Members

m_normal	Normal
m_thickness	Thickness of facet
m_vertIds	Vertex index
m_edgeIds	Edge index
m_userData	User data

Description

This is a facet structure that is created when the BVH type is specified during large mesh creation.

Structures

PfxExpandedTriMesh, PfxQuantizedTriMesh

Island mesh structure of array type

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxExpandedTriMesh {
        PfxUInt8 m_numVerts;
        PfxUInt8 m_numEdges;
        PfxUInt8 m_numFacets;
        PfxEdge *m_edges;
        PfxExpandedFacet *m_facets;
        PfxFloat3 *m_verts;
};
struct SCE_PFX_ALIGNED(16) PfxQuantizedTriMesh {
        PfxUInt8 m_numVerts;
        PfxUInt8 m_numEdges;
        PfxUInt8 m_numFacets;
        PfxEdge *m_edges;
        PfxQuantizedFacet *m_facets;
        PfxQuantize3 *m_verts;
};
```

Members

m_numVerts	Number of vertices
m_numEdges	Number of edges
m_numFacets	Number of triangles
m_edges	Pointer to edge buffer
m_facets	Pointer to facet buffer
m_verts	Pointer to vertex buffer

Description

This is an island mesh structure that is created when the array type is specified during large mesh creation. It expresses the partial meshes that make up large meshes and stores groups of divided triangles.

Island meshes being automatically created with the `pfxCreateLargeTriMesh()` function, they need not be explicitly created.

16-byte alignment is required.

Structures

PfxExpandedTriMeshBvh, PfxQuantizedTriMeshBvh

Island mesh structure of BVH type

Definition

```
#include <physics_effects/base_level/collision/pfx_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxExpandedTriMeshBvh {
        PfxUInt8 m_numVerts;
        PfxUInt8 m_numEdges;
        PfxUInt8 m_numFacets;
        PfxEdge *m_edges;
        PfxExpandedFacetBvh *m_facets;
        PfxFloat3 *m_verts;
        PfxAabb16 *m_bvhNodes;
        PfxUInt32 m_bvhRootId;
};
struct SCE_PFX_ALIGNED(16) PfxQuantizedTriMeshBvh {
        PfxUInt8 m_numVerts;
        PfxUInt8 m_numEdges;
        PfxUInt8 m_numFacets;
        PfxEdge *m_edges;
        PfxQuantizedFacetBvh *m_facets;
        PfxQuantize3 *m_verts;
        PfxAabb16 *m_bvhNodes;
        PfxUInt32 m_bvhRootId;
};
```

Members

m_numVerts	Number of vertices
m_numEdges	Number of edges
m_numFacets	Number of triangles
m_edges	Pointer to edge buffer
m_facets	Pointer to facet buffer
m_verts	Pointer to vertex buffer
m_bvhNodes	Pointer to node buffer of BVH
m_bvhRootId	Root node of BVH

Description

This is an island mesh structure that is created when the BVH type is specified during large mesh creation.

16-byte alignment is required.

Structures

PfxLargeTriMesh

Large mesh structure

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        PfxUInt32 m_type;
        PfxVector3 m_offset;
        PfxVector3 m_half;
        PfxUInt16 m_numIslands;
        PfxAabb16 *m_aabbList;
        PfxUInt32 m_numBvhNodes;
        PfxAabb16 *m_bvhNodes;
        PfxUInt32 m_bvhRootId;
        void *m_islands;
};
```

Members

m_type	Type
m_offset	Center of large mesh
m_half	Size of large mesh
m_numIslands	Number of islands
m_aabbList	AABB array of island mesh
m_numBvhNodes	Number of BVH nodes
m_bvhNodes	BVH node buffer
m_bvhRootId	Root node of BVH
m_islands	Array of island mesh

Description

This class expresses large mesh shapes. It is used to express complex shapes such as landscapes. One large mesh is made up of several island meshes. To create a large mesh, use the `pfxCreateLargeTriMesh()` utility function.

The following four patterns can be selected as types of large mesh during large mesh creation.

Type	Description
`SCE_PFX_LARGE_MESH_TYPE_EXPANDED_ARRAY`	The data is expanded and the island is stored in an array.
`SCE_PFX_LARGE_MESH_TYPE_QUANTIZED_ARRAY`	The data is compressed and the island is stored in an array.
`SCE_PFX_LARGE_MESH_TYPE_EXPANDED_BVH`	The data is expanded and the island is stored in a BVH.
`SCE_PFX_LARGE_MESH_TYPE_QUANTIZED_BVH`	The data is compressed and the island is stored in a BVH.

16-byte alignment is required.

Method List

Method	Description
`PfxLargeTriMesh`	Initializes the large mesh parameters
`getLocalPosition`	Gets the coordinate values of the local coordinates of the large mesh
`getWorldPosition`	Converts the local coordinates value of the large mesh to the original coordinates
`quantizePosition`	Compresses the coordinate values
`quantizeVector`	Compresses the vector
`quantizeNormal`	Compresses the normal vector
`quantizeFloat`	Compresses the floating-point value
`decodePosition`	Expands the compressed coordinate values
`decodeVector`	Expands the compressed vector
`decodeNormal`	Expands the compressed normal vector
`decodeFloat`	Expands the compressed floating-point value
`isUsingBvh`	Judges whether BVH is used for storing island
`isQuantized`	Judges whether mesh data is compressed

Constructors and Destructors

PfxLargeTriMesh

Initialize large mesh parameters

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct PfxLargeTriMesh {
        PfxLargeTriMesh();
};
```

Arguments

None

Return Values

None

Description

Initializes the large mesh parameters.

Public Methods

getLocalPosition

Get coordinate values for local coordinates of large mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxVecInt3 getLocalPosition(
              const PfxVector3 &worldPos
        ) const;
        inline PfxVecInt3 getLocalPosition(
              const PfxVector3 &worldMinPosition,
              const PfxVector3 &worldMaxPosition,
              PfxVecInt3 &localMinPosition,
              PfxVecInt3 &localMaxPosition
        ) const;
};
```

Arguments

worldPos	Coordinate value

Return Values

Returns the position in the local coordinate system of the large mesh as the `PfxVecInt3` type.

Description

This public method converts the coordinate values to the local coordinates of the large mesh. Locally, the coordinates are managed with integers.

getWorldPosition

Convert local coordinate values of large mesh to original coordinates

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxVector3 getWorldPosition(
             const PfxVecInt3 &localPos
        ) const;
};
```

Arguments

localPos	Local coordinates

Return Values

Returns the converted coordinate value

Description

The local coordinates are converted to integers and saved. Conversion to floating point numbers is done.

quantizePosition

Compress coordinate value

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxQuantize3 quantizePosition(
              const PfxVector3 &p
        ) const;
};
```

Arguments

p	Coordinate value

Return Values

Returns the compressed coordinate value.

Description

The coordinate value is quantized and converted into three 16-bit integer values.

quantizeVector

Compress vector

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxQuantize3 quantizeVector(
             const PfxVector3 &v
        ) const;
};
```

Arguments

v	Vector

Return Values

Returns the compressed vector.

Description

The vector is quantized and converted into three 16-bit integer values.

quantizeNormal

Compress normal vector

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxQuantize2 quantizeNormal(
             const PfxVector3 &n
        ) const;
};
```

Arguments

n	Normal vector

Return Values

Returns the compressed normal vector.

Description

The normal vector is quantized and converted into two 16-bit integer values.

quantizeFloat

Compress floating point value

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxQuantize quantizeFloat(
             const PfxVector3 &value
        ) const;
};
```

Arguments

value	Floating point value

Return Values

Returns the compressed floating point value.

Description

The floating point value is quantized and converted into a 16-bit integer value.

decodePosition

Expand compressed coordinate value

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxVector3 decodePosition(
              const PfxQuantize3 &q
        ) const;
};
```

Arguments

q	Compressed coordinate value

Return Values

Returns the expanded coordinate value.

Description

Converts the compressed coordinate value to the `PfxVector3` type.

decodeVector

Expand compressed vector

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxVector3 decodeVector(
             const PfxQuantize3 &q
        ) const;
};
```

Arguments

q	Compressed vector

Return Values

Returns the expanded vector.

Description

Converts the compressed vector to the `PfxVector3` type.

decodeNormal

Expand compressed normal vector

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxVector3 decodeNormal(
              const PfxQuantize2 &q
        ) const;
};
```

Arguments

q	Compressed normal vector

Return Values

Returns the expanded normal vector.

Description

Converts the compressed normal vector to the `PfxVector3` type.

decodeFloat

Expand compressed floating point value

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxFloat decodeFloat(
             PfxQuantize q
        ) const;
};
```

Arguments

q Compressed floating point value

Return Values

Returns the expanded floating point value.

Description

Converts the compressed floating point value to the `PfxFloat` type.

isUsingBvh

Judge if BVH is used to store island

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxBool isUsingBvh() const
};
```

Arguments

None

Return Values

Returns `true` when an island is stored as BVH.

isQuantized

Judge if mesh data is compressed

Definition

```
#include <physics_effects/base_level/collision/pfx_large_tri_mesh.h>
struct SCE_PFX_ALIGNED(16) PfxLargeTriMesh {
        inline PfxBool isQuantized() const
};
```

Arguments

None

Return Values

Returns `true` when mesh data is compressed.

Solver Body

Structures

PfxSolverBody

Solver body structure

Definition

```
#include <physics_effects/base_level/solver/pfx_solver_body.h>
struct SCE_PFX_ALIGNED(128) PfxSolverBody {
        PfxVector3 m_deltaLinearVelocity;
        PfxVector3 m_deltaAngularVelocity;
        PfxQuat m_orientation;
        PfxMatrix3 m_inertiaInv;
        PfxFloat m_massInv;
};
```

Members

m_deltaLinearVelocity	Linear velocity difference
m_deltaAngularVelocity	Angular velocity difference
m_orientation	Orientation
m_inertiaInv	Inverse matrix of inertia tensor
m_massInv	Inverse number of mass

Description

This is the solver body structure used for solver calculations. It sets the parameters required for solver body structures from the rigid body data with the `pfxSetupSolverBodies()` function.

128-byte alignment is required.

Joint

Structures

PfxJointConstraint

Joint constraint structure

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_constraint.h>
struct PfxJointConstraint {
        PfxInt8 m_lock;
        PfxInt8 m_warmStarting;
        PfxFloat m_movableLowerLimit;
        PfxFloat m_movableUpperLimit;
        PfxFloat m_bias;
        PfxFloat m_weight;
        PfxFloat m_damping;
        PfxFloat m_maxImpulse;
        PfxConstraintRow m_constraintRow;
};
```

Members

m_lock	Type of movement limit on constraint axis
m_warmStarting	Use result of previous frame as initial value
m_movableLowerLimit	Lower limit of movement range on constraint axis
m_movableUpperLimit	Upper limit of movement range on constraint axis
m_bias	Adjustment of constraint force in return direction if movement range is exceeded
m_weight	Weight value for adjusting effect of calculated constraint force
m_damping	Damping value expressing friction on constraint axis
m_maxImpulse	Maximum impulse value
m_constraintRow	Internal parameter used for constraint calculation

Description

This structure expresses the joint constraint.

PfxJoint

Joint structure

Definition

```
#include <physics_effects/base_level/solver/pfx_joint.h>
struct SCE_PFX_ALIGNED(128) PfxJoint {
        PfxUInt8 m_active;
        PfxUInt8 m_numConstraints;
        PfxUInt8 m_type;
        PfxUInt16 m_rigidBodyIdA;
        PfxUInt16 m_rigidBodyIdB;
        PfxJointConstraint m_constraints[6];
        void *m_userData;
        PfxVector3 m_param[4];
        PfxVector3 m_anchorA;
        PfxVector3 m_anchorB;
        PfxMatrix3 m_frameA;
        PfxMatrix3 m_frameB;
};
```

Members

m_active	Joint active
m_numConstraints	Number of constraints to be used
m_type	Type of joint
m_rigidBodyIdA	Index of parent rigid body
m_rigidBodyIdB	Index of child rigid body
m_constraints	Constraint
m_userData	User data
m_param	Internally used parameter
m_anchorA	Connection point in parent rigid body coordinates
m_anchorB	Connection point in child rigid body coordinates
m_frameA	Joint frame in parent rigid body coordinates
m_frameB	Joint frame in child rigid body coordinates

Description

This is a structure that expresses a joint. It has a data structure that forms the basis of all joints. 128-byte alignment is required.

Sub Data of Collision

Structures

PfxSubData

Structure storing sub data of large mesh

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxUInt8 m_type;
        struct {
              PfxUInt32 userData;
              PfxUInt16 islandId;
              PfxUInt8 facetId;
              PfxUInt16 s;
              PfxUInt16 t;
        } m_facetLocal;
};
```

Members

m_type	Type
m_facetLocal.userData	User data
m_facetLocal.islandId	Island index
m_facetLocal.facetId	Triangle index
m_facetLocal.s	Coordinates on triangle in barycentric coordinate system
m_facetLocal.t	Coordinates on triangle in barycentric coordinate system

Description

This structure is used to get the accurate large mesh part, during collision judgment or raycast.

By using the barycentric coordinate system, it is possible to calculate the contact point on a triangle with the following equation.

When the vertices that make up the triangle are P0, P1, and P2, the coordinates can be calculated from the values of *s* and *t* with the following equation.

Coordinate = *s* * (P1 - P0) + *t* * (P2 - P0)

The values set for type are as follows.

Value	Description
PfxSubData::NONE	No value is set to the sub data
PfxSubData::MESH_INFO	The large mesh data is saved to the sub data

Method List

Method	Description
`PfxSubData`	Sets the parameters to their initial values
`getIslandId`	Gets the island index
`setIslandId`	Sets the island index
`getFacetId`	Gets the triangle index
`setFacetId`	Sets the triangle index
`getFacetLocalS`, `getFacetLocalT`	Gets the coordinates on the triangle in the barycentric coordinate system
`setFacetLocalS`, `setFacetLocalT`	Sets the coordinates on the triangle in the barycentric coordinate system
`getUserData`	Gets the user data
`setUserData`	Sets the user data

Constructors and Destructors

PfxSubData

Set parameters to their initial values

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxSubData();
};
```

Arguments

None

Return Values

None

Description

Initializes the parameters of the sub data.

Public Methods

getIslandId

Get island index

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxUInt16 getIslandId();
};
```

Arguments

None

Return Values

Returns the island index.

Description

This public method gets the island index of the large mesh.

setIslandId

Set island index

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        void setIslandId(PfxUInt16 i);
};
```

Arguments

i	Island index

Return Values

None

Description

This public method sets the island index of the large mesh.

getFacetId

Get triangle index

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxUInt8 getFacetId();
};
```

Arguments

None

Return Values

Returns the triangle index.

Description

This public method gets the triangle index of the island mesh.

setFacetId

Set triangle index

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        void setFacetId(PfxUInt8 i);
};
```

Arguments

i	Triangle index

Return Values

None

Description

This public method sets the triangle index of the island mesh.

getFacetLocalS, getFacetLocalT

Get coordinate values on triangle in barycentric coordinate system

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxFloat getFacetLocalS();
        PfxFloat getFacetLocalT();
};
```

Arguments

None

Return Values

Returns the coordinate values on the triangle in the barycentric coordinate system.

Description

This public method gets the coordinate values on the triangle in the barycentric coordinate system. Values `s` and `t` are parameters for calculating the coordinates on the triangle.

setFacetLocalS, setFacetLocalT

Set coordinate values on triangle in barycentric coordinate system

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        void setFacetLocalS(PfxFloat s);
        void setFacetLocalT(PfxFloat t);
};
```

Arguments

s	Value *s* on triangle in barycentric coordinate system
t	Value *t* on triangle in barycentric coordinate system

Return Values

None

Description

This public method sets the coordinate values on the triangle in the barycentric coordinate system.

getUserData

Get user data

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        PfxUInt32 getUserData();
};
```

Arguments

None

Return Values

Returns the user data.

Description

This public method gets the user data.

setUserData

Set user data

Definition

```
#include <physics_effects/base_level/collision/pfx_sub_data.h>
struct PfxSubData {
        void setUserData(PfxUInt32 data);
};
```

Arguments

data	User data

Return Values

None

Description

This public method sets the user data.

Collision Data

Structures

PfxContactPoint

Structure that stores contact point

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
struct PfxContactPoint {
        PfxUInt8 m_duration;
        PfxUInt8 m_shapeIdA;
        PfxUInt8 m_shapeIdB;
        PfxSubData m_subData;
        PfxFloat m_distance;
        PfxFloat m_localPointA[3];
        PfxFloat m_localPointB[3];
        PfxConstraintRow m_constraintRow[3];
};
```

Members

m_duration	Continuous count
m_shapeIdA	Index of shape A
m_shapeIdB	Index of shape B
m_subData	Sub data of collision
m_distance	Penetration depth
m_localPointA	Collision coordinates in coordinate system of shape A
m_localPointB	Collision coordinates in coordinate system of shape B
m_constraintRow	Structure storing constraint data for solver operation

Description

This is a structure that stores the parameters related to one contact point. The continuous count is incremented as long as the collision is maintained.

In case of collision with a large mesh, the information indicating the part where the collision occurred is stored in *m_subData*.

Classes

PfxContactManifold

Class that stores collision data

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {};
```

Description

This is the class that manages the collision data. The contact point coordinates change moment by moment as the result of the movement of rigid bodies, but the validity of contact points can be updated by calling `refresh()` at every frame. Contact points whose collision coordinates are at a certain distance or greater are automatically discarded. In addition, the continuous count is incremented while the collision is maintained.

128-byte alignment is required.

Method List

Method	Description
`reset`	Resets the parameters to their initial values
`addContactPoint`	Adds a contact point
`removeContactPoint`	Deletes a contact point
`getNumContacts`	Gets the number of contact points
`getContactPoint`	Gets a contact point
`refresh`	Updates all the contact points
`merge`	Merges separate collision data
`getDuration`	Gets the continuous count
`getRigidBodyIdA`	Gets the index of rigid body A
`getRigidBodyIdB`	Gets the index of rigid body B

Public Methods

reset

Reset parameters to their initial values

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        void reset(
              PfxUInt16 rigidBodyIdA,
              PfxUInt16 rigidBodyIdB
        );
};
```

Arguments

rigidBodyIdA	Index A of rigid body
rigidBodyIdB	Index B of rigid body

Return Values

None

Description

This public method initializes the collision data parameters.

addContactPoint

Add contact point

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        void addContactPoint(
              PfxFloat newDistance,
              const PfxVector3 &newNormal,
              const PfxPoint3 &newPointA,
              const PfxPoint3 &newPointB,
              PfxSubData subData
        );
        void addContactPoint(
              const PfxContactPoint &cp
        );
};
```

Arguments

newDistance	Penetration depth
newNormal	Collision normal in world coordinate system
newPointA	Collision coordinates in rigid body A local coordinate system
newPointB	Collision coordinates in rigid body B local coordinate system
subData	Sub data
cp	Structure that stores contact points

Return Values

None

Description

This public method adds contact points. Up to four contact points can be held. When a fifth contact point is added, the four points defining the largest area and including the contact point with the deepest depth are selected.

removeContactPoint

Remove contact point

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        void removeContactPoint(int i);
};
```

Arguments

i	Index of contact point

Return Values

None

Description

This public method removes contact points.

getNumContacts

Get number of contact points

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        int getNumContacts() const;
};
```

Arguments

None

Return Values

Returns the number of contact points.

Description

This public method gets the number of contact points.

getContactPoint

Get contact point

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        PfxContactPoint &getContactPoint(int i);
        const PfxContactPoint &getContactPoint(int i) const;
};
```

Arguments

i	Index of contact point

Return Values

Returns a contact point.

Description

This public method gets the contact point of the specified index.

refresh

Update all contact points

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        void refresh(
              const PfxVector3 &pA,
              const PfxQuat &qA,
              const PfxVector3 &pB,
              const PfxQuat &qB
        );
};
```

Arguments

pA	Position of rigid body A
qA	Orientation of rigid body A
pB	Position of rigid body B
qB	Orientation of rigid body B

Return Values

None

Description

This public method updates all the contact points from the rigid body's new position and orientation. Contact points at a certain distance or greater are discarded.

merge

Merge separate collision data

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        void merge(const PfxContactManifold &contact)
};
```

Arguments

contact	Collision data

Return Values

None

Description

This public method merges separate collision data.

getDuration

Get continuous count

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        PfxUInt16 getDuration() const
};
```

Arguments

None

Return Values

Returns the continuous count.

Description

This public method gets the continuous count.

getRigidBodyIdA, getRigidBodyIdB

Get indices of rigid bodies A and B

Definition

```
#include <physics_effects/base_level/collision/pfx_contact_manifold.h>
class SCE_PFX_ALIGNED(128) PfxContactManifold {
        PfxUInt16 getRigidBodyIdA() const;
        PfxUInt16 getRigidBodyIdB() const;
};
```

Arguments

None

Return Values

Returns the number of contact points.

Description

This public method gets the number of contact points.

Ray

Structures

PfxRayInput

Ray input structure

Definition

```
#include <physics_effects/low_level/collision/pfx_ray.h>
struct SCE_PFX_ALIGNED(16) PfxRayInput {
        PfxVector3 m_startPosition;
        PfxVector3 m_direction;
        PfxUInt32 m_contactFilterSelf;
        PfxUInt32 m_contactFilterTarget;
        PfxUInt8 m_facetMode;
};
```

Members

m_startPosition	Start point of ray
m_direction	Direction vector of ray
m_contactFilterSelf	Self contact filter (Default value: 0xffffffff)
m_contactFilterTarget	Target contact filter (Default value: 0xffffffff)
m_facetMode	Intersect mode (Default value: `SCE_PFX_RAY_FACET_MODE_FRONT_ONLY`)

Description

This ray structure is expressed with a start point and a direction. It is specified so that the magnitude of the *m_direction* direction vector is the same as the length of the ray.

16-byte alignment is required.

The contact filter is a filter that allows individual specification of whether to perform judgment of intersection with rigid bodies and shapes.

m_facetMode is a flag that specifies whether to take into consideration the front and back of the triangle when judging intersection with a large mesh.

For *m_facetMode*, set one of the following values. If a value other than the following values is specified, intersection judgment is not performed.

Value	Description
`SCE_PFX_RAY_FACET_MODE_FRONT_ONLY`	Judgment only when ray intersects from front to back
`SCE_PFX_RAY_FACET_MODE_BACK_ONLY`	Judgment only when ray intersects from back to front
`SCE_PFX_RAY_FACET_MODE_FRONT_AND_BACK`	Both front and back used for judgment

PfxRayOutput

Ray output structure

Definition

```
#include <physics_effects/low_level/collision/pfx_ray.h>
struct SCE_PFX_ALIGNED(16) PfxRayOutput {
        PfxVector3 m_contactPoint;
        PfxVector3 m_contactNormal;
        PfxFloat m_variable;
        PfxUInt16 m_objectId;
        PfxUInt8 m_shapeId;
        PfxBool m_contactFlag : 1;
        PfxSubData m_subData;
};
```

Members

m_contactPoint	Intersection coordinates in world coordinate system
m_contactNormal	Normal vector in world coordinate system
m_variable	Variable indicating intersection position on ray
m_objectId	Object index
m_shapeId	Shape index
m_contactFlag	Intersection flag
m_subData	Sub data

Description

This structure receives the output from the raycast. The result of the intersection nearest the start point of the ray is stored. Upon detection of the intersection of a ray and object, *m_contactFlag* is set to true, the rigid body index is returned to *m_objectId*, and the index indicating which number the shape is among the rigid body shapes is returned to *m_shapeId*. Moreover, in the case of intersection with a large mesh, the data indicating which part the intersection occurred at is stored in *m_subData*.

16-byte alignment is required.

Simulation Island

Structures

PfxIslandUnit

Simulation island unit structure

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
struct PfxIslandUnit {
        // Member not disclosed
};
```

Members

Members are not disclosed

Description

This unit expresses a single object (rigid body) included in a simulation island.

PfxIsland

Simulation island structure

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
struct PfxIsland {
        // Member not disclosed
};
```

Members

Members are not disclosed

Description

This is a structure that controls simulation islands.

Task Manager

Classes

PfxPsp2TaskManager

PlayStation®Vita task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {};
```

Description

This is a task management class for executing simulation processing in parallel. It is specified for the argument of an API that supports this class, and is executed in parallel on multiple threads by dividing the processing into multiple tasks. Threads that are started up from the task manager immediately enter the event wait status, and the processing waits until an event is issued from the API.

Method List

Method	Description
`PfxPsp2TaskManager`	Constructor of task manager
`~PfxPsp2TaskManager`	Destructor of task manager
`getWorkBytes`	Gets the size of the work area
`getNumTasks`	Gets the number of tasks to be used
`setNumTasks`	Sets the number of tasks to be used
`initialize`	Initializes the task manager
`finalize`	Terminates the task manager

Constructors and Destructors

PfxPsp2TaskManager

Constructor of task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        PfxPsp2TaskManager(
              PfxUInt32 numTasks,
              PfxUInt32 maxTasks,
              void *workBuff,
              PfxUInt32 workBytes
        );
};
```

Arguments

numTasks	Number of execution tasks
maxTasks	Maximum number of tasks
workBuff	Pointer to work area
workBytes	Size of work area

Return Values

None

Description

Allocate the value returned from `getWorkBytes()` as the size of the work area buffer to be used by the task manager, and specify it to the constructor.

~PfxPsp2TaskManager

Destructor of task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        ~PfxPsp2TaskManager();
};
```

Arguments

None

Return Values

None

Description

Clears the internal buffer.

Public Methods

getWorkBytes

Get size of work area to be used by task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        static PfxUInt32 getWorkBytes(PfxUInt32 maxTasks);
};
```

Arguments

maxTasks	Maximum number of tasks

Return Values

Returns the size of the work area to be used by the task manager.

Description

This public method gets the size of the work area to be used by the task manager. Allocate a work buffer of the size corresponding to the returned value, and specify it as the constructor argument of the task manager.

getNumTasks

Get number of execution tasks

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        PfxUInt32 getNumTasks() const;
};
```

Arguments

None

Return Values

Returns the number of execution tasks.

Description

This public method gets the number of execution tasks.

setNumTasks

Set number of execution tasks

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        void setNumTasks(PfxUInt32 tasks);
};
```

Arguments

tasks	Number of execution tasks

Return Values

None

Description

This public method sets the number of execution tasks. If the maximum number of tasks is exceeded, the maximum number of tasks is set.

initialize

Initialize task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        void initialize();
};
```

Arguments

None

Return Values

None

Description

This public method initializes the task manager. The number of threads corresponding to the maximum number of tasks is started up.

finalize

Terminate task manager

Definition

```
#include <physics_effects/low_level/task/pfx_task_manager.psp2.h>
class PfxPsp2TaskManager {
        void finalize();
};
```

Arguments

None

Return Values

None

Description

This public method terminates the task manager. It closes all the threads and releases their resources.

Heap Manager

Classes

PfxHeapManager

Heap manager

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {};
```

Description

Heap Manager is a simple memory manager that uses stacks. The memory allocated by an application beforehand is specified to the constructor of the heap manager. The heap manager pools this memory and uses it by carving out some memory from this pool when needed. Memory allocation and release being implemented by using stacks, there is the limitation of having to release memory resources in the opposite sequence to the allocation sequence, but this operation is extremely fast and memory fragmentation is not an issue.

Method List

Method	Description
`PfxHeapManager`	Constructor of heap manager
`~PfxHeapManager`	Destructor of heap manager
`allocate`	Allocates buffers
`deallocate`	Releases buffer allocations
`clear`	Releases all buffers
`getAllocated`	Gets the number of bytes of allocated buffers
`getRest`	Gets the number of remaining bytes that can be allocated

Constructors and Destructors

PfxHeapManager

Constructor of heap manager

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        PfxHeapManager(
              PfxUInt8 *buf,
              PfxInt32 bytes
        );
};
```

Arguments

buf	Address of buffer
bytes	Size of buffer

Return Values

None

Description

The buffers assigned to the constructor argument are initialized through internal pooling.

~PfxHeapManager

Destructor of heap manager

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        ~PfxHeapManager();
};
```

Arguments

None

Return Values

None

Description

Nothing is done.

Public Methods

allocate

Allocate memory of specified size

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        void *allocate(
              size_t bytes,
              PfxInt32 alignment
        )
};
```

Arguments

bytes	Number of bytes of memory to be allocated
alignment	Alignment specification

Return Values

Returns the address of the allocated memory.

Description

The minimum allocation size is 16 bytes. The alignments that can be specified are only 16-byte and 128-byte alignment. If nothing is specified, 16-byte alignment is automatically selected. Assert is called if allocation fails.

deallocate

Release allocated memory

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        void deallocate(void *p);
};
```

Arguments

p	Address of allocated memory

Return Values

None

Description

This public method releases allocated memory. Be sure to call this public method in the reverse sequence to the call sequence.

clear

Initialize memory

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        void clear();
};
```

Arguments

None

Return Values

None

Description

This public method releases all the allocated memories.

getAllocated

Get number of bytes of allocated memory

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        PfxInt32 getAllocated();
};
```

Arguments

None

Return Values

Returns the number of bytes of the allocated memory.

Description

This public method gets the number of bytes of the allocated memory.

getRest

Get number of bytes of allocatable memory

Definition

```
#include <physics_effects/base_level/base/pfx_heap_manager.h>
class PfxHeapManager {
        PfxInt32 getRest();
};
```

Arguments

None

Return Values

Returns the number of bytes of the allocatable memory.

Description

This public method gets the number of bytes of the allocatable memory.

Joint APIs

pfxUpdateJointPairs

Update joint pairs

Definition

```
#include <physics_effects/base_level/solver/pfx_joint.h>
void pfxUpdateJointPairs(
        PfxConstraintPair &pair,
        PfxUInt32 jointId,
        const PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB
)
```

Arguments

pair	Joint pair
jointId	Joint index
joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state

Return Values

None

Description

Sets joint-related parameters to one `PfxConstraintPair` structure. One pair has a data structure that allows efficient access to related data and is used during solver operations.

PfxBallJointInitParam

Ball joint initialization parameter

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_ball.h>
struct PfxBallJointInitParam {
        PfxVector3 anchorPoint;
        PfxMatrix3 jointFrame;
        PfxBool enableJointFrame;
        PfxFloat damping;
};
```

Members

anchorPoint	Connection coordinate value of joint (Default value: (0.0f, 0.0f, 0.0f))
jointFrame	Joint frame
enableJointFrame	Enable joint frame (default: `false`)
damping	Damping

Description

This structure is used to specify parameters when initializing ball joints with `pfxInitializeBallJoint()`. It is used to reproduce a joint connected at one point only that can turn freely.

pfxInitializeBallJoint

Initialize ball joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_ball.h>
PfxInt32 pfxInitializeBallJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxBallJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameters

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This is a function that initializes ball joints.

PfxSwingTwistJointInitParam

Swing twist joint initialization parameters

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_swing_twist.h>
struct PfxSwingTwistJointInitParam {
        PfxVector3 anchorPoint;
        PfxVector3 twistAxis;
        PfxFloat twistLowerAngle;
        PfxFloat twistUpperAngle;
        PfxFloat swingLowerAngle;
        PfxFloat swingUpperAngle;
        PfxMatrix3 jointFrame;
        PfxBool enableJointFrame;
        PfxFloat damping;
        PfxFloat bias;
};
```

Members

anchorPoint	Connection coordinate value of joint (default value: (0.0f, 0.0f,0 .0f))
twistAxis	Twist axis (default value: (1.0f, 0.0f, 0.0f))
twistLowerAngle	Radian value of lower limit of twist angle (default value: -0.26f)
twistUpperAngle	Radian value of upper limit of twist angle (default value: 0.26f)
swingLowerAngle	Radian value of lower limit of swing angle (default value: 0.0f)
swingUpperAngle	Radian value of upper limit of swing angle (default value: 0.7f)
jointFrame	Joint frame
enableJointFrame	Enable joint frame (default value: `false`)
damping	Damping
bias	Adjustment value for the joint's rigidity

Description

This structure is used to specify parameters when initializing swing twist joints with `pfxInitializeSwingTwistJoint()`. It is used when reproducing joints that have the movement range of cylinders like shoulders. Specify that the twist angle is between -180 degrees and 180 degrees, and the swing angle is between 0 degrees and 180 degrees.

pfxInitializeSwingTwistJoint

Initialize swing twist joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_swing_twist.h>
PfxInt32 pfxInitializeSwingTwistJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxSwingTwistJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This function initializes swing twist joints.

PfxHingeJointInitParam

Hinge joint initialization parameters

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_hinge.h>
struct PfxHingeJointInitParam {
        PfxVector3 anchorPoint;
        PfxVector3 axis;
        PfxFloat lowerAngle;
        PfxFloat upperAngle;
        PfxMatrix3 jointFrame;
        PfxBool enableJointFrame;
        PfxFloat damping;
        PfxFloat bias;
};
```

Members

anchorPoint	Connection coordinate value of joint (default value: (0.0f, 0.0f, 0.0f))
axis	Movable axis of hinge (default value: (1.0f, 0.0f, 0.0f))
lowerAngle	Radian value of lower limit of hinge angle (default value: 0.0f)
upperAngle	Radian value of upper limit of hinge angle (default value: 0.0f)
jointFrame	Joint frame
enableJointFrame	Enable joint frame (default value: `false`)
damping	Damping
bias	Adjustment value for the joint's rigidity

Description

This structure is used to specify parameters when initializing hinge joints with `pfxInitializeHingeJoint()`. It is used when reproducing joints that have a hinge-like structure. Specify that the hinge angle is between -180 degrees and 180 degrees.

pfxInitializeHingeJoint

Initialize hinge joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_hinge.h>
PfxInt32 pfxInitializeHingeJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxHingeJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This function initializes hinge joints.

PfxSliderJointInitParam

Slider joint initialization parameter

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_slider.h>
struct PfxSliderJointInitParam {
        PfxVector3 anchorPoint;
        PfxVector3 direction;
        PfxFloat lowerDistance;
        PfxFloat upperDistance;
        PfxMatrix3 jointFrame;
        PfxBool enableJointFrame;
        PfxFloat damping;
        PfxFloat bias;
};
```

Members

anchorPoint	Connection coordinate value of joint (default value: (0.0f, 0.0f, 0.0f))
direction	Movable axis of slider (default value: (1.0f, 0.0f, 0.0f))
lowerDistance	Lower limit of movable distance (default value: 0.0f)
upperDistance	Upper limit of movable distance (default value: 0.0f)
jointFrame	Joint frame
enableJointFrame	Enable joint frame (default value: `false`)
damping	Damping
bias	Adjustment value for the joint's rigidity

Description

This structure is used to specify parameters when initializing slider joints with `pfxInitializeSliderJoint()`. It is used to reproduce joints that can move only in a specific direction.

pfxInitializeSliderJoint

Initialize slider joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_slider.h>
PfxInt32 pfxInitializeSliderJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxSliderJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This function initializes slider joints.

PfxFixJointInitParam

Fixed joint initialization parameter

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_fix.h>
struct PfxFixJointInitParam {
        PfxVector3 anchorPoint;
};
```

Members

anchorPoint	Connection coordinate value of joint (default value: (0.0f, 0.0f, 0.0f))

Description

This structure is used to specify parameters when initializing fixed joints with `pfxInitializeFixJoint()`. It is used to fix two rigid bodies at a single point.

pfxInitializeFixJoint

Initialize fixed joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_fix.h>
PfxInt32 pfxInitializeFixJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxFixJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This function initializes fixed joints.

PfxUniversalJointInitParam

Universal joint initialization parameter

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_universal.h>
struct PfxUniversalJointInitParam {
        PfxVector3 anchorPoint;
        PfxVector3 axis;
        PfxVector3 upVec;
        PfxFloat swing1LowerAngle;
        PfxFloat swing1UpperAngle;
        PfxFloat swing2LowerAngle;
        PfxFloat swing2UpperAngle;
        PfxMatrix3 jointFrame;
        PfxBool enableJointFrame;
        PfxFloat damping;
        PfxFloat bias;
};
```

Members

anchorPoint	Connection coordinate value of joint (default value: (0.0f, 0.0f, 0.0f))
axis	Movable axis (default value: (1.0f, 0.0f, 0.0f))
upVec	Up vector (default value: (0.0f, 1.0f, 0.0f))
swing1LowerAngle	Lower limit of swing 1 angle (default value: -0.7f)
swing1UpperAngle	Upper limit of swing 1 angle (default value: 0.7f)
swing2LowerAngle	Lower limit of swing 2 angle (default value: -0.7f)
swing2UpperAngle	Upper limit of swing 2 angle (default value: 0.7f)
jointFrame	Joint frame
enableJointFrame	Enable joint frame (default value: `false`)
damping	Damping
bias	Adjustment value of the joint's rigidity

Description

This structure is used to specify parameters when initializing universal joints with `pfxInitializeUniversalJoint()`. It is used to reproduce a joint that has two swing axes. Specify that the swing angle is between 0 degrees and 180 degrees.

When the movable scopes of both axes exceed 90 degrees and the two axes become closer to being perpendicular, behavior will become unstable. When the movable scope of one axis exceeds 90 degrees, make sure to specify a value that does not exceed 90 for the other axis.

pfxInitializeUniversalJoint

Initialize universal joint

Definition

```
#include <physics_effects/base_level/solver/pfx_joint_universal.h>
PfxInt32 pfxInitializeUniversalJoint(
        PfxJoint &joint,
        const PfxRigidState &stateA,
        const PfxRigidState &stateB,
        const PfxUniversalJointInitParam &param
)
```

Arguments

joint	Joint structure
stateA	Parent rigid body state
stateB	Child rigid body state
param	Joint initialization parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is incorrect.

Description

This function initializes universal joints.

Broad Phase APIs

pfxUpdateBroadphaseProxy

Create broadphase proxy

Definition

```
#include
<physics_effects/base_level/broadphase/pfx_update_broadphase_proxy.h>
PfxInt32 pfxUpdateBroadphaseProxy(
        PfxBroadphaseProxy &proxy,
        const PfxRigidState &state,
        const PfxCollidable &coll,
        const PfxVector3 &worldCenter,
        const PfxVector3 &worldExtent,
        PfxUInt32 axis
)
```

Arguments

proxy	Broadphase proxy
state	Rigid body state
coll	Rigid body shape
worldCenter	Center of world space
worldExtent	Size of world space
axis	Detection axis

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_WORLD` when the position of the rigid body is outside the world space.

Description

Copies the parameters from the rigid body data to the broadphase proxy. The broadphase proxy is used to detect intersecting pairs. The minimum value on the detection axis of the bounding box of the rigid body is stored as the sort key.

pfxUpdateBroadphaseProxy

Creation of broadphase proxy

Definition

```
#include
<physics_effects/base_level/broadphase/pfx_update_broadphase_proxy.h>
PfxInt32 pfxUpdateBroadphaseProxy(
        PfxBroadphaseProxy &proxyX,
        PfxBroadphaseProxy &proxyY,
        PfxBroadphaseProxy &proxyZ,
        PfxBroadphaseProxy &proxyXb,
        PfxBroadphaseProxy &proxyYb,
        PfxBroadphaseProxy &proxyZb,
        const PfxRigidState &state,
        const PfxCollidable &coll,
        const PfxVector3 &worldCenter,
        const PfxVector3 &worldExtent
)
```

Arguments

proxyX	Broadphase proxy (sort in +X axis direction)
proxyY	Broadphase proxy (sort in +Y axis direction)
proxyZ	Broadphase proxy (sort in +Z axis direction)
proxyXb	Broadphase proxy (sort in -X axis direction)
proxyYb	Broadphase proxy (sort in -Y axis direction)
proxyZb	Broadphase proxy (sort in -Z axis direction)
state	Rigid body state
coll	Rigid body shape
worldCenter	Center of world space
worldExtent	Size of world space

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_WORLD` when the position of the rigid body is outside the world space.

Description

Simultaneously creates broadphase proxies corresponding to each of the axes from the rigid body data. The minimum (or maximum) value of the bounding box of the rigid body for each of the axes is stored as the sort key.

PfxUpdateBroadphaseProxiesParam

Parameters used to update broadphase proxy arrays

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxUpdateBroadphaseProxiesParam {
        void *workBuff;
        PfxUInt32 workBytes;
        PfxBroadphaseProxy *proxiesX;
        PfxBroadphaseProxy *proxiesY;
        PfxBroadphaseProxy *proxiesZ;
        PfxBroadphaseProxy *proxiesXb;
        PfxBroadphaseProxy *proxiesYb;
        PfxBroadphaseProxy *proxiesZb;
        PfxRigidState *offsetRigidStates;
        PfxCollidable *offsetCollidables;
        PfxUInt32 numRigidBodies;
        PfxUInt32 outOfWorldBehavior;
        PfxVector3 worldCenter;
        PfxVector3 worldExtent;
};
```

Members

workBuff	Pointer to work area
workBytes	Size of work area
proxiesX	Broadphase proxy array sorted in +X axis direction (16-byte alignment)
proxiesY	Broadphase proxy array sorted in +Y axis direction (16-byte alignment)
proxiesZ	Broadphase proxy array sorted in +Z axis direction (16-byte alignment)
proxiesXb	Broadphase proxy array sorted in -X axis direction (16-byte alignment)
proxiesYb	Broadphase proxy array sorted in -Y axis direction (16-byte alignment)
proxiesZb	Broadphase proxy array sorted in -Z axis direction (16-byte alignment)
offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetCollidables	Offset address of rigid body shape array (128-byte alignment)
numRigidBodies	Number of broadphase proxies
outOfWorldBehavior	Handling of rigid body placed out of world (default value: 0)
worldCenter	Center of world space
worldExtent	Size of world space

Description

This structure is used to specify the parameters passed to the `pfxUpdateBroadphaseProxies()` function for updating broadphase proxy arrays. Six broadphase proxy arrays, corresponding to the number of rigid bodies, must be allocated in advance.

The handling of rigid bodies placed out of world can be specified with the following flag combinations.

Value	Description
`SCE_PFX_OUT_OF_WORLD_BEHAVIOR_FIX_MOTION`	Changes the motion type to fixed
`SCE_PFX_OUT_OF_WORLD_BEHAVIOR_REMOVE_PROXY`	Removes proxy array member from proxy array

PfxUpdateBroadphaseProxiesResult

Structure that receives broadphase proxy array update result

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxUpdateBroadphaseProxiesResult {
        PfxInt32 numOutOfWorldProxies;
};
```

Members

numOutOfWorldProxies	Number of rigid bodies placed out of world

Description

This structure is used to receive the result of the `pfxUpdateBroadphaseProxies()` function for updating broadphase proxy arrays.

pfxGetWorkBytesOfUpdateBroadphaseProxies

Get size of work area used for updating broadphase proxy array

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxUInt32 pfxGetWorkBytesOfUpdateBroadphaseProxies(
        PfxUInt32 numRigidBodies
)
PfxUInt32 pfxGetWorkBytesOfUpdateBroadphaseProxies(
        PfxUInt32 numRigidBodies,
        PfxUInt32 maxTasks
)
```

Arguments

numRigidBodies	Number of rigid bodies
maxTasks	Number of tasks

Return Values

Returns the size of the work area used for broadphase proxy creation.

Description

Allocate a work buffer of the returned size and specify it for the `PfxUpdateBroadphaseProxiesParam` broadphase proxy array update parameter.

pfxUpdateBroadphaseProxies

Update broadphase proxy array

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxInt32 pfxUpdateBroadphaseProxies(
        PfxUpdateBroadphaseProxiesParam &param,
        PfxUpdateBroadphaseProxiesResult &result
)
// Parallel version
PfxInt32 pfxUpdateBroadphaseProxies(
        PfxUpdateBroadphaseProxiesParam &param,
        PfxUpdateBroadphaseProxiesResult &result,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
result	Structure that receives result
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid
`SCE_PFX_ERR_OUT_OF_BUFFER`	Specified buffer capacity is insufficient

Description

This structure is used to update all the broadphase proxy arrays corresponding to all the axis directions, and sort them by axis. Updated broadphase proxy arrays can be used for intersecting pair detection and raycasts.

If a rigid body has been placed outside the specified world area, its handling is decided according to the specification of the *outOfWorldBehavior* flag of `PfxUpdateBroadphaseProxiesParam`. If removal from the proxy array is specified, the final number of valid arrays is `param.`*numRigidBodies* `- result.`*numOutOfWorldProxies*.

PfxFindPairsParam

Parameters used for intersecting pair detection

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxFindPairsParam {
        void *workBuff;
        PfxUInt32 workBytes;
        void *pairBuff;
        PfxUInt32 pairBytes;
        PfxBroadphaseProxy *proxies;
        PfxUInt32 numProxies;
        PfxUInt32 maxPairs;
        int axis;
};
```

Members

workBuff	Pointer to work area
workBytes	Size of work area
pairBuff	Pointer to pair buffer (16-byte alignment)
pairBytes	Size of pair buffer
proxies	Pointer to broadphase proxy array (16-byte alignment)
numProxies	Number of broadphase proxies
maxPairs	Maximum number of pairs that can be acquired
axis	Detection axis

Description

This structure is used to specify the parameters to be passed to the `pfxFindPairs()` intersecting pair detection function. Align the pair buffer at the 16-byte boundary. The broadphase proxy array must be sorted in advance along the detection axis.

PfxFindPairsResult

Structure that receives intersecting pair detection result

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxFindPairsResult {
        PfxBroadphasePair *pairs;
        PfxUInt32 numPairs;
};
```

Members

pairs	Pair array
numPairs	Number of detected pairs

Description

This structure is used to receive the results of the `pfxFindPairs()` intersecting pair detection function. Detected intersecting pairs are sorted according to a unique value created from two rigid body indices.

pfxGetWorkBytesOfFindPairs

Get size of work area used for intersecting pair detection

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxUInt32 pfxGetWorkBytesOfFindPairs(
        PfxUInt32 maxPairs,
        PfxUInt32 maxTasks
)
```

Arguments

maxPairs	Maximum number of pairs that can be acquired
maxTasks	Number of tasks (default value: 1)

Return Values

Returns the size of the work area used for intersecting pair detection.

Description

Allocate a work buffer of the returned size and specify it for the `PfxFindPairsParam` intersecting pair detection parameter.

pfxGetPairBytesOfFindPairs

Get size of pair buffer returned upon intersecting pair detection

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxUInt32 pfxGetPairBytesOfFindPairs(
        PfxUInt32 maxPairs
)
```

Arguments

maxPairs	Maximum number of pairs that can be acquired

Return Values

Returns the size of the pair buffer returned upon intersecting pair detection.

Description

Allocate a pair buffer of the returned size and specify it for the `PfxFindPairsParam` intersecting pair detection parameter.

pfxFindPairs

Detect intersection of broadphase proxies

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxInt32 pfxFindPairs(
        PfxFindPairsParam &param,
        PfxFindPairsResult &result
)
// Parallel version
PfxInt32 pfxFindPairs(
        PfxFindPairsParam &param,
        PfxFindPairsResult &result,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
result	Structure that receives result
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid
`SCE_PFX_ERR_OUT_OF_BUFFER`	Specified buffer capacity is insufficient
`SCE_PFX_ERR_OUT_OF_MAX_PAIRS`	Number of detected pairs exceeds specified value

Description

Detects pairs of broadphase proxies whose bounding boxes (AABB) intersect, and returns the result to `PfxFindPairsResult`. The area of the pair array returned as a result is acquired from the pair buffer specified to `PfxFindPairsParam`. Do not discard the pair buffer until after it has finished being used.

Since it is not possible to know in advance the maximum number of detected pairs, specify the maximum number of pairs with some extra. If the number of detected pairs exceeds the maximum number of pairs, the `SCE_PFX_ERR_OUT_OF_MAX_PAIRS` error is returned. In this case, allocate a larger capacity buffer and execute `pfxFindPairs()` again.

* To create a broadphase proxy, use `pfxUpdateBroadphaseProxy()`.

PfxDecomposePairsParam

Parameters used for comparing intersecting pairs

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxDecomposePairsParam {
        void *workBuff;
        PfxUInt32 workBytes;
        void *pairBuff;
        PfxUInt32 pairBytes;
        PfxBroadphasePair *previousPairs;
        PfxUInt32 numPreviousPairs;
        PfxBroadphasePair *currentPairs;
        PfxUInt32 numCurrentPairs;
};
```

Members

workBuff	Pointer to work area
workBytes	Size of work area
pairBuff	Pointer to pair buffer(16-byte alignment)
pairBytes	Size of pair buffer
previousPairs	Sorted previous pair array (16-byte alignment)
numPreviousPairs	Number of previous pairs
currentPairs	Sorted newly detected pair array (16-byte alignment)
numCurrentPairs	Number of newly detected pairs

Description

This structure is used to specify the parameters to be passed to the `pfxDecomposePairs()` function for comparing two pair arrays. Sort in advance the pair arrays specified for *previousPairs* and *currentPairs*. The key value for sorting is automatically appended by the `pfxFindPairs()` function. Align the pair buffer and pair arrays at the 16-byte boundary.

PfxDecomposePairsResult

Structure that receives result of intersecting pair comparison

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
struct PfxDecomposePairsResult {
        PfxBroadphasePair *outNewPairs;
        PfxUInt32 numOutNewPairs;
        PfxBroadphasePair *outKeepPairs;
        PfxUInt32 numOutKeepPairs;
        PfxBroadphasePair *outRemovePairs;
        PfxUInt32 numOutRemovePairs;
};
```

Members

outNewPairs	New pair array
numOutNewPairs	Number of new pairs
outKeepPairs	Kept pair array
numOutKeepPairs	Number of kept pairs
outRemovePairs	Discarded pair array
numOutRemovePairs	Number of discarded pairs

Description

This structure is used to receive the result of the `pfxDecomposePairs()` intersecting pair comparison function.

pfxGetWorkBytesOfDecomposePairs

Get size of work area used for intersecting pair comparison

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxUInt32 pfxGetWorkBytesOfDecomposePairs(
        PfxUInt32 numPreviousPairs,
        PfxUInt32 numCurrentPairs,
        int maxTasks
)
```

Arguments

numPreviousPairs	Number of previous detected pairs
numCurrentPairs	Number of current detected pairs
maxTasks	Number of tasks (default value: 1)

Return Values

Returns the size of the work area used for intersecting pair comparison.

Description

Allocate a work buffer of the returned size and specify it for the `PfxDecomposePairsParam` intersecting pair comparison parameter.

pfxGetPairBytesOfDecomposePairs

Get size of pair buffer returned during intersecting pair comparison

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxUInt32 pfxGetPairBytesOfDecomposePairs(
         PfxUInt32 numPreviousPairs,
         PfxUInt32 numCurrentPairs
)
```

Arguments

numPreviousPairs	Number of previous detected pairs
numCurrentPairs	Number of current detected pairs

Return Values

Returns the size of the pair buffer returned during intersecting pair comparison.

Description

Allocate a pair buffer of the returned size and specify it to the `PfxDecomposePairsParam` intersecting pair comparison parameter.

pfxDecomposePairs

Compare two intersecting pairs

Definition

```
#include <physics_effects/low_level/broadphase/pfx_broadphase.h>
PfxInt32 pfxDecomposePairs(
        PfxDecomposePairsParam &param,
        PfxDecomposePairsResult &result
)
// Parallel version
PfxInt32 pfxDecomposePairs(
        PfxDecomposePairsParam &param,
        PfxDecomposePairsResult &result,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
result	Structure that receives result
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
SCE_PFX_ERR_INVALID_VALUE	Input parameter is invalid
SCE_PFX_ERR_INVALID_ALIGN	Input parameter alignment is invalid
SCE_PFX_ERR_OUT_OF_BUFFER	Specified buffer capacity is insufficient

Description

Compares new and old pair arrays and returns the result to `PfxDecomposePairsResult`. Returns the result as new pairs in the case of pairs that only exist in the new pair arrays, as kept pairs in the case of pairs that exist in both new and old pair arrays, and as discard pairs in the case of pairs that only exist in the old pair arrays.

The area of the pair array returned as the result is acquired from the pair buffer specified to `PfxDecomposePairsParam`. Do not discard the pair buffer until after it has finished being used.

Collision Detection APIs

PfxDetectCollisionParam

Parameters used for collision detection

Definition

```
#include <physics_effects/low_level/collision/pfx_collision_detection.h>
struct PfxDetectCollisionParam {
        PfxConstraintPair *contactPairs;
        PfxUInt32 numContactPairs;
        PfxContactManifold *offsetContactManifolds;
        PfxRigidState *offsetRigidStates;
        PfxCollidable *offsetCollidables;
        PfxUInt32 numRigidBodies;
};
```

Members

contactPairs	Collision detection pair array (16-byte alignment)
numContactPairs	Number of collision detection pairs
offsetContactManifolds	Offset address of contact data array (128-byte alignment)
offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetCollidables	Offset address of rigid body shape array (128-byte alignment)
numRigidBodies	Total number of rigid bodies

Description

This structure is used to specify the parameters required for collision detection. The index of the pair contact data output in advance from the broad phase is set and specified as the collision detection pair.

pfxDetectCollision

Detect collision of rigid bodies

Definition

```
#include <physics_effects/low_level/collision/pfx_collision_detection.h>
PfxInt32 pfxDetectCollision(
        PfxDetectCollisionParam &param
)
// Parallel version
PfxInt32 pfxDetectCollision(
        PfxDetectCollisionParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Gets the index to the rigid bodies and collision data from the collision detection pair array, and perform access to the offset address + index as the actual data. If the data is not valid, assert is executed.

Gets the position in the world and judgment state from the two rigid body states included in the pair and the rigid body shapes, and calls a different judgment function according to each shape combination. If a collision is detected, the collision coordinates, the repelling direction and the penetration depth are calculated, and up to four contact points is stored to the contact data.

Constraint Solver APIs

PfxSetupSolverBodiesParam

Parameters used for solver body setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
struct PfxSetupSolverBodiesParam {
        PfxRigidState *states;
        PfxRigidBody *bodies;
        PfxSolverBody *solverBodies;
        PfxUInt32 numRigidBodies;
};
```

Members

states	Rigid body state array (128-byte alignment)
bodies	Rigid body attribute array (128-byte alignment)
solverBodies	Solver body array (128-byte alignment)
numRigidBodies	Total number of rigid bodies

Description

Updates the solver bodies based on the rigid body data. The same number of solver body arrays as the number of rigid bodies must be prepared and allocated in advance.

pfxSetupSolverBodies

Solver body setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
PfxInt32 pfxSetupSolverBodies(
        PfxSetupSolverBodiesParam &param
)
// Parallel version
PfxInt32 pfxSetupSolverBodies(
        PfxSetupSolverBodiesParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Executes the setup of the solver body structure to be used for solver operations. The parameters required for the operations are copied from the body data, and the solver body is used instead of rigid body states or rigid body attributes in the operation loop.

PfxSetupContactConstraintsParam

Parameters used for collision constraint setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
struct PfxSetupContactConstraintsParam {
        PfxConstraintPair *contactPairs;
        PfxUInt32 numContactPairs;
        PfxContactManifold *offsetContactManifolds;
        PfxRigidState *offsetRigidStates;
        PfxRigidBody *offsetRigidBodies;
        PfxSolverBody *offsetSolverBodies;
        PfxUInt32 numRigidBodies;
        PfxFloat timeStep;
        PfxFloat separateBias;
};
```

Members

contactPairs	Collision pair array (16-byte alignment)
numContactPairs	Number of collision pairs
offsetContactManifolds	Offset address of contact data array (128-byte alignment)
offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetRigidBodies	Offset address of rigid body attribute array (128-byte alignment)
offsetSolverBodies	Offset address of solver body array (128-byte alignment)
numRigidBodies	Total number of rigid bodies
timeStep	Timestep (default value: 0.016f)
separateBias	Bias value (default value: 0.2f)

Description

This structure is used to specify the parameters required for collision constraint setup. The pair array output from the collision judgment is specified as the collision pair. The index to the `PfxContactManifold` collision data must be allocated for all the pairs.

The time is specified in units of 1 second for the timestep. The bias value is the adjustment value of the constraint force pulling apart rigid bodies that have collided. Specify it in the range of 0.0 to 1.0.

pfxSetupContactConstraints

Collision constraint setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
PfxInt32 pfxSetupContactConstraints(
        PfxSetupContactConstraintsParam &param
)
// Parallel version
PfxInt32 pfxSetupContactConstraints(
        PfxSetupContactConstraintsParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Sets up the data to be used for solver operations, based on the rigid body and collision data.

PfxSetupJointConstraintsParam

Parameters used for joint constraint setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
struct PfxSetupJointConstraintsParam {
        PfxConstraintPair *jointPairs;
        PfxUInt32 numJointPairs;
        PfxJoint *offsetJoints;
        PfxRigidState *offsetRigidStates;
        PfxRigidBody *offsetRigidBodies;
        PfxSolverBody *offsetSolverBodies;
        PfxUInt32 numRigidBodies;
        PfxFloat timeStep;
};
```

Members

jointPairs	Joint pair array (16-byte alignment)
numJointPairs	Number of joint pairs
offsetJoints	Offset address of joint array (128-byte alignment)
offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetRigidBodies	Offset address of rigid body attribute array (128-byte alignment)
offsetSolverBodies	Offset address of solver body array
numRigidBodies	Total number of rigid bodies
timeStep	Timestep (default value: 0.016f)

Description

This structure is used to specify the parameters required for joint constraint setup. The joint pairs must be prepared in advance from the joint structure using the `pfxUpdateJointPairs()` function.

The time is specified in units of 1 second for the timestep.

pfxSetupJointConstraints

Joint constraint setup

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
PfxInt32 pfxSetupJointConstraints(
        PfxSetupJointConstraintsParam &param
)
// Parallel version
PfxInt32 pfxSetupJointConstraints(
        PfxSetupJointConstraintsParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Sets up the data to be used for solver operations, based on the rigid body and joint data.

PfxSolveConstraintsParam

Parameters used for constraint solver

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
struct PfxSolveConstraintsParam {
        void *workBuff;
        PfxUInt32 workBytes;
        PfxConstraintPair *contactPairs;
        PfxUInt32 numContactPairs;
        PfxContactManifold *offsetContactManifolds;
        PfxConstraintPair *jointPairs;
        PfxUInt32 numJointPairs;
        PfxJoint *offsetJoints;
        PfxRigidState *offsetRigidStates;
        PfxSolverBody *offsetSolverBodies;
        PfxUInt32 numRigidBodies;
        PfxUInt32 iteration;
};
```

Members

workBuff	Pointer to work area
workBytes	Size of work area
contactPairs	Collision pair array (16-byte alignment)
numContactPairs	Number of collision pairs
offsetContactManifolds	Offset address of contact data array (128-byte alignment)
jointPairs	Joint pair array (16-byte alignment)
numJointPairs	Number of joint pairs
offsetJoints	Offset address of joint array (128-byte alignment)
offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetSolverBodies	Offset address of solver body array (128-byte alignment)
numRigidBodies	Total number of rigid bodies
iteration	Number of operation iterations (default value: 5)

Description

This structure is used to specify the parameters to be passed to the `pfxSolveConstraints()` constraint solver function. In order to simultaneously solve the collision and joint constraints, operations are repeated for all the constraints for the specified number of iterations. The larger the number of iterations, the higher the accuracy is, but at the price of lower performance.

pfxGetWorkBytesOfSolveConstraints

Get size of work area used for constraint solver

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
PfxUInt32 pfxGetWorkBytesOfSolveConstraints(
        PfxUInt32 numRigidBodies,
        PfxUInt32 numContactPairs,
        PfxUInt32 numJointPairs
)
```

Arguments

numRigidBodies	Total number of rigid bodies
numContactPairs	Number of collision pairs
numJointPairs	Number of joint pairs

Return Values

Returns the size of the work area to be used for the constraint solver.

Description

Allocate the work buffer of the returned size and specify it for the `PfxSolveConstraintsParam` constraint solver parameter.

pfxSolveConstraints

Constraint solver

Definition

```
#include <physics_effects/low_level/solver/pfx_constraint_solver.h>
PfxInt32 pfxSolveConstraints(
        PfxSolveConstraintsParam &param
)
// Parallel version
PfxInt32 pfxSolveConstraints(
        PfxSolveConstraintsParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

This is a solver function that simultaneously solves multiple constraints caused by collisions and joints between rigid bodies, using a reiterative operation solving process. The constraint force output from the solver is converted to velocity and is finally reflected to the rigid body state.

External Force and Position Calculation APIs

pfxApplyExternalForce

Apply external force to rigid body

Definition

```
#include <physics_effects/base_level/solver/pfx_integrate.h>
static SCE_PFX_FORCE_INLINE void pfxApplyExternalForce(
        PfxRigidState &state,
        const PfxRigidBody &body,
        const PfxVector3 &extForce,
        const PfxVector3 &extTorque,
        PfxFloat timeStep
)
```

Arguments

state	Rigid body state
body	Rigid body attribute
extForce	External force
extTorque	External torque
timeStep	Timestep

Return Values

None

Description

The given external force is converted to the velocity and reflected to the rigid body state. The time is specified in units of 1 second for the timestep.

pfxIntegrate

Update position of rigid body

Definition

```
#include <physics_effects/base_level/solver/pfx_integrate.h>
static SCE_PFX_FORCE_INLINE void pfxIntegrate(
        PfxRigidState &state,
        const PfxRigidBody &body,
        PfxFloat timeStep
)
```

Arguments

state	Rigid body state
body	Rigid body attribute
timeStep	Timestep

Return Values

None

Description

Calculates the position after the timestep from the velocity of the rigid body and reflects it to the rigid body state. The velocity is maintained as is.

To perform batch processing to arrays of rigid bodies, use `pfxUpdateRigidStates()`.

PfxUpdateRigidStatesParam

Parameters used for rigid body position update

Definition

```
#include <physics_effects/low_level/solver/pfx_update_rigid_states.h>
struct PfxUpdateRigidStatesParam {
        PfxRigidState *states;
        PfxRigidBody *bodies;
        PfxUInt32 numRigidBodies;
        PfxFloat timeStep;
};
```

Members

states	Rigid body state array (128-byte alignment)
bodies	Rigid body attribute array (128-byte alignment)
numRigidBodies	Total number of rigid bodies
timeStep	Timestep

Description

This structure is used to specify the parameters to be passed to the `pfxUpdateRigidStates()` function for executing position update for an array of rigid bodies. The time is specified in units of 1 second for the timestep.

pfxUpdateRigidStates

Update rigid body position

Definition

```
#include <physics_effects/low_level/solver/pfx_update_rigid_states.h>
PfxInt32 pfxUpdateRigidStates(
        PfxUpdateRigidStatesParam &param
)
// Parallel version
PfxInt32 pfxUpdateRigidStates(
        PfxUpdateRigidStatesParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Updates in batch the positions of rigid bodies.

Simulation Island APIs

PfxGenerateIslandParam

Parameters used for simulation island creation

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
struct PfxGenerateIslandParam {
        void *islandBuff;
        PfxUInt32 islandBytes;
        PfxConstraintPair *pairs;
        PfxUInt32 numPairs;
        PfxUInt32 numObjects;
};
```

Members

islandBuff	Pointer to island buffer
islandBytes	Size of island buffer
pairs	Pair array (16-byte alignment)
numPairs	Number of pairs
numObjects	Number of objects (rigid bodies)

Description

This structure is used to specify the parameters to be passed to the `pfxGenerateIsland()` function for creating simulation islands. Get the buffer size for the simulation island output with `pfxGetIslandBytesOfGenerateIsland()` beforehand, and set the allocated buffer.

Align the pair array to the 16-byte boundary.

PfxGenerateIslandResult

Structure that receives simulation island result

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
struct PfxGenerateIslandResult {
        PfxIsland *island;
};
```

Members

island	Simulation island structure

Description

This structure is used to store the result returned from the `pfxGenerateIsland()` function for creating simulation island.

pfxGetIslandBytesOfGenerateIsland

Get island buffer size to be used for simulation island creation

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxUInt32 pfxGetIslandBytesOfGenerateIsland(
        PfxUInt32 numObjects
)
```

Arguments

numObjects	Total number of objects

Return Values

Returns the island buffer size to be used for simulation island creation.

Description

Allocate a buffer of the returned size and specify it as the `PfxGenerateIslandParam` input parameter for the simulation island creation function.

pfxGenerateIsland

Create simulation island

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxInt32 pfxGenerateIsland(
        PfxGenerateIslandParam &param,
        PfxGenerateIslandResult &result
)
```

Arguments

param	Input parameter
result	Structure that receives result

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid
`SCE_PFX_ERR_OUT_OF_BUFFER`	Specified buffer capacity is insufficient

Description

Builds a simulation island from a pair array. Do not discard the simulation island buffer specified in the input parameter until after simulation island use has finished.

pfxAppendPairs

Add pairs to simulation island

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxUInt32 pfxAppendPairs(
        PfxIsland *island,
        PfxConstraintPair *pairs,
        PfxUInt32 numPairs
)
```

Arguments

island	Simulation island structure (16-byte alignment)
pairs	Pair array (16-byte alignment)
numPairs	Number of pairs

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	Input parameter is invalid
`SCE_PFX_ERR_INVALID_ALIGN`	Input parameter alignment is invalid

Description

Updates an already created simulation island by adding a pair array. The size of the buffer that is used for the simulation island does not change through this operation.

If the argument value is invalid, an assert is called.

pfxGetNumIslands

Get total number of simulation islands

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxUInt32 pfxGetNumIslands(
        const PfxIsland *islands
)
```

Arguments

islands	Simulation island structure

Return Values

Returns the total number of simulation islands.

Description

Gets the total number of simulation islands.

If the argument value is invalid, an assert is called.

pfxGetFirstUnitInIsland

Get first unit of specified simulation island

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxIslandUnit *pfxGetFirstUnitInIsland(
        const PfxIsland *islands,
        PfxUInt32 islandId
)
```

Arguments

islands	Simulation island structure
islandId	Index of simulation island

Return Values

Returns the first unit belonging to the specified simulation island.

If an out-of-range value is set, assert is called.

Description

Gets the first unit of the specified simulation island.

If the argument value is invalid, an assert is called.

pfxGetNextUnitInIsland

Get next unit

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxIslandUnit *pfxGetNextUnitInIsland(
        const PfxIslandUnit *islandUnit
)
```

Arguments

islandUnit	Simulation island unit structure

Return Values

Returns the next unit. Returns NULL if there is no next unit.

Description

Gets the next unit.

If the argument value is invalid, an assert is called.

pfxGetUnitId

Gets unit index

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxUInt32 pfxGetUnitId(
        const PfxIslandUnit *islandUnit
)
```

Arguments

islandUnit	Simulation island unit structure

Return Values

Returns the index of the specified unit. Assert is called if the unit does not exist.

Description

Gets the unit index. This index can be used as the index of the rigid body.

If the argument value is invalid, an assert is called.

pfxGetIslandId

Get index of island that stores unit

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
PfxUInt32 pfxGetIslandId(
        const PfxIsland *islands,
        PfxUInt32 unitId
)
```

Arguments

islands	Simulation island structure
unitId	Index of simulation island unit

Return Values

Returns index of island to which unit belongs. If the unit is not found, assert is called.

Description

Gets the index of the island that stores the unit.

If the argument value is invalid, an assert is called.

pfxResetIsland

Reset island

Definition

```
#include <physics_effects/low_level/collision/pfx_island_generation.h>
void pfxResetIsland(
        PfxIsland *islands
)
```

Arguments

islands	Simulation island structure

Return Values

None

Description

Clears the contents of the simulation island.

If the argument value is invalid, an assert is called.

Raycast APIs

PfxRayCastParam

Parameters used for raycast

Definition

```
#include <physics_effects/low_level/collision/pfx_ray_cast.h>
struct PfxRayCastParam {
        PfxRigidState *offsetRigidStates;
        PfxCollidable *offsetCollidables;
        PfxBroadphaseProxy *proxiesX;
        PfxBroadphaseProxy *proxiesY;
        PfxBroadphaseProxy *proxiesZ;
        PfxBroadphaseProxy *proxiesXb;
        PfxBroadphaseProxy *proxiesYb;
        PfxBroadphaseProxy *proxiesZb;
        PfxUInt32 numProxies;
        PfxVector3 rangeCenter;
        PfxVector3 rangeExtent;
};
```

Members

offsetRigidStates	Offset address of rigid body state array (128-byte alignment)
offsetCollidables	Offset address of rigid body shape array (128-byte alignment)
proxiesX	Broadphase proxy array sorted in +X axis direction (16-byte alignment)
proxiesY	Broadphase proxy array sorted in +Y axis direction (16-byte alignment)
proxiesZ	Broadphase proxy array sorted in +Z axis direction (16-byte alignment)
proxiesXb	Broadphase proxy array sorted in -X axis direction (16-byte alignment)
proxiesYb	Broadphase proxy array sorted in -Y axis direction (16-byte alignment)
proxiesZb	Broadphase proxy array sorted in -Z axis direction (16-byte alignment)
numProxies	Number of broadphase proxies (= Total number of rigid bodies)
rangeCenter	Center of area to be raycast
rangeExtent	Size of area to be raycast

Description

Creates broadphase proxy of all rigid bodies to be judged with the `pfxUpdateBroadphaseProxy()` function and allocates it to parameters. The optimum broadphase proxy for the ray direction is selected by providing broadphase proxies for all the axes.

Allocate as is the world size set during broadphase proxy creation as the raycast area.

pfxCastSingleRay

Single raycast

Definition

```
#include <physics_effects/low_level/collision/pfx_ray_cast.h>
void pfxCastSingleRay(
        const PfxRayInput &rayInput,
        PfxRayOutput &rayOutput,
        const PfxRayCastParam &param
)
```

Arguments

rayInput	Ray input structure
rayOutput	Ray output structure
param	Input parameter

Return Values

There are no return values. If an invalid input parameter is detected, assert is called.

Description

One ray is cast onto the world. The intersection data that is the closest to the start point is stored to the `PfxRayOutput` structure.

When the `pfxCastSingleRay()` function is called from multiple threads for the same world space, do not update the input parameter contents until all the raycasts have been completed.

pfxCastRays

Batch raycast

Definition

```
#include <physics_effects/low_level/solver/pfx_batched_ray_cast.h>
void pfxCastRays(
        PfxRayInput *rayInputs,
        PfxRayOutput *rayOutputs,
        int numRays,
        PfxRayCastParam &param
)
// Parallel version
void pfxCastRays(
        PfxRayInput *rayInputs,
        PfxRayOutput *rayOutputs,
        int numRays,
        PfxRayCastParam &param,
        PfxTaskManager *taskManager
)
```

Arguments

rayInputs	Array of ray input structures (16-byte alignment)
rayOutputs	Array of ray output structures (16-byte alignment)
numRays	Total number of rays
param	Input parameter
taskManager	Task manager (only when multithread is used)

Return Values

There are no return values. If *param* is an invalid value, an assert is called.

Description

Casts multiple rays onto the world space. Ray input array processing is done sequentially, and upon detection of an intersection, the intersection information closest to the start point is stored to the corresponding `PfxRayOutput` structure.

Sort APIs

pfxParallelSort

Sort

Definition

```
#include <physics_effects/low_level/sort/pfx_parallel_sort.h>
PfxInt32 pfxParallelSort(
        PfxSortData16 *data,
        PfxUInt32 numData,
        void *workBuff,
        PfxUInt32 workBytes
)
PfxInt32 pfxParallelSort(
        PfxSortData32 *data,
        PfxUInt32 numData,
        void *workBuff,
        PfxUInt32 workBytes
)
// Parallel version
PfxInt32 pfxParallelSort(
        PfxSortData16 *data,
        PfxUInt32 numData,
        void *workBuff,
        PfxUInt32 workBytes,
        PfxTaskManager *taskManager
)
PfxInt32 pfxParallelSort(
        PfxSortData32 *data,
        PfxUInt32 numData,
        void *workBuff,
        PfxUInt32 workBytes,
        PfxTaskManager *taskManager
)
```

Arguments

data	Array of data to be sorted (16-byte alignment)
numData	Number of data to be sorted
workBuff	Pointer to work area (16-byte alignment)
workBytes	Size of work area
taskManager	Task manager (only when multithread is used)

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_OUT_OF_BUFFER`	The size of the work area is insufficient
`SCE_PFX_ERR_INVALID_ALIGN`	The alignment of the pointer to the work area is invalid

Description

Sorts an array of sort data that is 16-byte or 32-byte aligned. Used for pair or broadphase proxy array sorting.

Utility APIs

pfxCalcMassBox

Calculate box mass

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxFloat pfxCalcMassBox(
        PfxFloat density,
        const PfxVector3 &halfExtent
)
```

Arguments

density	Density
halfExtent	Size of box

Return Values

Returns the mass.

Description

Calculates the mass from the size and density of the box.

pfxCalcInertiaBox

Calculate inertia tensor of box

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxMatrix3 pfxCalcInertiaBox(
        const PfxVector3 &halfExtent,
        PfxFloat mass
)
```

Arguments

halfExtent	Size of box
mass	Mass

Return Values

Returns the inertia tensor.

Description

Calculates the inertia tensor from the size and mass of the box.

pfxCalcMassSphere

Calculate mass of sphere

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxFloat pfxCalcMassSphere(
        PfxFloat density,
        PfxFloat radius
)
```

Arguments

density	Density
radius	Radius of sphere

Return Values

Returns the mass.

Description

Calculates the mass from the radius and density of the sphere.

pfxCalcInertiaSphere

Calculate inertia tensor of sphere

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxMatrix3 pfxCalcInertiaSphere(
        PfxFloat radius,
        PfxFloat mass
)
```

Arguments

radius	Radius of sphere
mass	Mass

Return Values

Returns the inertia tensor.

Description

Calculates the inertia tensor from the radius and mass of the sphere.

pfxCalcMassCylinder

Calculate mass of cylinder

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxFloat pfxCalcMassCylinder(
        PfxFloat density,
        PfxFloat halfLength,
        PfxFloat radius
)
```

Arguments

density	Density
halfLength	Length of cylinder
radius	Radius of cylinder

Return Values

Returns the mass.

Description

Calculates the mass from the length, radius and density of the cylinder.

pfxCalcInertiaCylinderX, pfxCalcInertiaCylinderY, pfxCalcInertiaCylinderZ

Calculate inertia tensor of cylinder

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxMatrix3 pfxCalcInertiaCylinderX(
        PfxFloat halfLength,
        PfxFloat radius,
        PfxFloat mass
)
PfxMatrix3 pfxCalcInertiaCylinderY(
        PfxFloat halfLength,
        PfxFloat radius,
        PfxFloat mass
)
PfxMatrix3 pfxCalcInertiaCylinderZ(
        PfxFloat halfLength,
        PfxFloat radius,
        PfxFloat mass
)
```

Arguments

halfLength	Length of cylinder
radius	Radius of cylinder
mass	Mass

Return Values

Returns the inertia tensor.

Description

Calculates the inertia tensor from the length, radius and mass of the cylinder.

Use one of the functions depending on the followings.

- When cylinder axis runs parallel to X axis, `pfxCalcInertiaCylinderX()`
- When cylinder axis runs parallel to Y axis, `pfxCalcInertiaCylinderY()`
- When cylinder axis runs parallel to Z axis, `pfxCalcInertiaCylinderZ()`

pfxMassTranslate

Translate inertia tensor

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxMatrix3 pfxMassTranslate(
        PfxFloat mass,
        const PfxMatrix3 &inertia,
        const PfxVector3 &translation
)
```

Arguments

mass	Mass
inertia	Inertia tensor
translation	Translation

Return Values

Returns the inertia tensor after translation.

Description

The position of the center of gravity can be changed by translating the inertia tensor.

pfxMassRotate

Rotate inertia tensor

Definition

```
#include <physics_effects/util/pfx_mass.h>
PfxMatrix3 pfxMassRotate(
        const PfxMatrix3 &inertia,
        const PfxMatrix3 &rotate
)
```

Arguments

inertia	Inertia tensor
rotate	Rotation matrix

Return Values

Returns the inertia tensor following rotation.

Description

Rotates the inertia tensor.

PfxCreateConvexMeshParam

Parameters used for convex mesh creation function

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
struct PfxCreateConvexMeshParam {
        PfxUInt32 flag;
        PfxFloat *verts;
        PfxUInt32 numVerts;
        void *triangles;
        PfxUInt32 numTriangles;
        PfxUInt32 vertexStrideBytes;
        PfxUInt32 triangleStrideBytes;
};
```

Members

flag	Flag (default value: `SCE_PFX_MESH_FLAG_16BIT_INDEX` | `SCE_PFX_MESH_FLAG_AUTO_ELIMINATION`)
verts	Pointer to vertex buffer
numVerts	Number of vertices
triangles	Pointer to triangle buffer
numTriangles	Number of triangles
vertexStrideBytes	Number of bytes between vertex data (default value: 12)
triangleStrideBytes	Number of bytes between triangle data (default value: 6)

Description

This structure is used to specify parameters when creating a convex mesh with the `pfxCreateConvexMesh()` function.

The following combinations of values are input to *flag*.

Value	Description
`SCE_PFX_MESH_FLAG_NORMAL_FLIP`	Inverts triangle index sequence
`SCE_PFX_MESH_FLAG_16BIT_INDEX`	Makes triangle index a 16-bit integer (`unsigned`)
`SCE_PFX_MESH_FLAG_32BIT_INDEX`	Makes triangle index a 32-bit integer (`unsigned`)
`SCE_PFX_MESH_FLAG_AUTO_ELIMINATION`	Deletes triangles with same vertex and surface area of 0

pfxCreateConvexMesh

Create convex mesh

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
PfxInt32 pfxCreateConvexMesh(
        PfxConvexMesh &convex,
        const PfxCreateConvexMeshParam &param
)
```

Arguments

convex	Convex mesh
param	Input parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	The number of vertices or triangles is 0 or NULL has been specified for the pointer
`SCE_PFX_ERR_OUT_OF_RANGE`	The number of vertices or triangles exceeds the designated number
`SCE_PFX_ERR_INVALID_FLAG`	The flag setting is incorrect

Description

Creates a convex mesh from the vertices and triangle buffers.

Whether the input shape is a convex shape is not judged within the function.

The buffer for storing the vertices and triangles when creating convex meshes is allocated dynamically.

After use, always release with `pfxReleaseConvexMesh()`.

Up to 128 vertices or 64 triangles can be held by a convex mesh. If the designated value is exceeded, the `SCE_PFX_ERR_OUT_OF_RANGE` error is returned.

pfxReleaseConvexMesh

Release convex mesh

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
void pfxReleaseConvexMesh(
        PfxConvexMesh &cmesh
)
```

Arguments

cmesh	Convex mesh

Return Values

None

Description

Releases the buffer allocated with `pfxCreateConvexMesh()`.

PfxCreateLargeTriMeshParam

Parameters used for large mesh creation

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
struct PfxCreateLargeTriMeshParam {
        PfxUInt32 flag;
        PfxFloat *verts;
        PfxUInt32 numVerts;
        void *triangles;
        PfxUInt32 *userData;
        PfxUInt32 numTriangles;
        PfxUInt32 vertexStrideBytes;
        PfxUInt32 triangleStrideBytes;
        PfxUInt32 numFacetsLimit;
        PfxFloat islandsRatio;
        PfxFloat defaultThickness;
};
```

Members

flag	Flag (default value: `SCE_PFX_MESH_FLAG_16BIT_INDEX`| `SCE_PFX_MESH_FLAG_AUTO_ELIMINATION`)
verts	Pointer to vertex buffer
numVerts	Number of vertices
triangles	Pointer to triangle buffer
userData	Pointer to user data buffer (array of 32-bit integer values)
numTriangles	Number of triangles
vertexStrideBytes	Number of bytes between vertex data (default value: 12)
triangleStrideBytes	Number of bytes between triangle data (default value: 6)
numFacetsLimit	Threshold of number of triangles registered to island (default value: 15)
islandsRatio	Size ratio of islands for entire mesh (default value: 0.2f)
defaultThickness	Default value of triangle thickness (default value: 0.025f)

Description

This structure is used to specify the parameters for creating a large mesh with the `pfxCreateLargeTriMesh()` function.

The following combinations of values are input to *flag*.

Value	Description
`SCE_PFX_MESH_FLAG_NORMAL_FLIP`	Inverts triangle index sequence
`SCE_PFX_MESH_FLAG_16BIT_INDEX`	Makes triangle index a 16-bit integer (`unsigned`)
`SCE_PFX_MESH_FLAG_32BIT_INDEX`	Makes triangle index a 32-bit integer (`unsigned`)
`SCE_PFX_MESH_FLAG_AUTO_ELIMINATION`	Deletes triangles with same vertex and surface area of 0
`SCE_PFX_MESH_FLAG_AUTO_THICKNESS`	Adds thickness to triangle
`SCE_PFX_MESH_FLAG_USE_BVH`	Uses BVH structure for storing a mesh
`SCE_PFX_MESH_FLAG_USE_QUANTIZED`	Quantizes mesh data for compression purposes
`SCE_PFX_MESH_FLAG_OUTPUT_INFO`	Outputs messages when creating a mesh
`SCE_PFX_MESH_FLAG_HIGH_QUALITY`	Optimizes the mesh structure to output

pfxCreateLargeTriMesh

Create large mesh

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
PfxInt32 pfxCreateLargeTriMesh(
        PfxLargeTriMesh &lmesh,
        const PfxCreateLargeTriMeshParam &param
)
```

Arguments

lmesh	Large mesh
param	Input parameter

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns one of the following error codes (negative value) for errors.

Value	Description
`SCE_PFX_ERR_INVALID_VALUE`	The number of vertices or triangles is 0 or NULL has been specified for the pointer.
`SCE_PFX_ERR_OUT_OF_RANGE`	The number of input parameter vertices or triangles exceeds the designated number.
`SCE_PFX_ERR_OUT_OF_BUFFER`	Buffer could not be allocated due to insufficient memory.
`SCE_PFX_ERR_INVALID_FLAG`	The flag setting is invalid.
`SCE_PFX_ERR_OUT_OF_RANGE_VERTEX`	The number of vertices exceeds the maximum number that can be included in one island.
`SCE_PFX_ERR_OUT_OF_RANGE_EDGE`	The number of edges exceeds the maximum number that can be included in one island.
`SCE_PFX_ERR_OUT_OF_RANGE_FACET`	The number of triangles exceeds the maximum number that can be included in one island.
`SCE_PFX_ERR_OUT_OF_RANGE_ISLAND`	The number of islands exceeds the maximum number that can be included in a large mesh.
`SCE_PFX_ERR_ZERO_AREA_FACET`	Detects a triangle whose surface area after compression will be 0

Description

Creates a large mesh from the vertices and triangle buffers.

The mesh structure can be efficiently searched hierarchically by dividing the input triangle into multiple groups (islands). The maximum number of islands that can be held by a large mesh is 512. Up to 128 vertices and 64 triangles can be held by an island. If the designated value is exceeded, the `SCE_PFX_ERR_OUT_OF_RANGE` error is returned. Arbitrary user data (32-bit integer values) can be assigned to individual triangles. The buffer for storing the islands or additional data during large mesh creation is allocated dynamically.

Following use, be sure to release the allocated buffer with the `pfxReleaseLargeTriMesh()` function.

Although the edge shared by three or more triangles does not result in an error, a warning message will be output, and the third or later triangles will be excluded from the judgment.

pfxReleaseLargeTriMesh

Release large mesh

Definition

```
#include <physics_effects/util/pfx_mesh_creator.h>
void pfxReleaseLargeTriMesh(
        PfxLargeTriMesh &lmesh
)
```

Arguments

lmesh	Large mesh

Return Values

None

Description

Releases the allocated buffer with the `pfxCreateLargeTriMesh()` function.

pfxSetUtilityAllocator

Specify memory allocator used by utility function group

Definition

```
#include <physics_effects/util/pfx_util_common.h>
void pfxSetUtilityAllocator(
        SCE_PFX_FUNC_ALLOC func_alloc,
        SCE_PFX_FUNC_REALLOC func_realloc,
        SCE_PFX_FUNC_FREE func_free
)
```

Arguments

func_alloc	Pointer to the memory allocator function
func_realloc	Pointer to the memory reallocator function
func_free	Pointer to the memory release function

Return Values

None

Description

Sets the memory allocator called by a utility function to the one designated by the user.

If the allocator is not specified with this function, the standard functions `memalign()`, `reallocalign()` and `free()` are used for the utility.

The types of the function pointer are as follows.

```
typedef void* (*SCE_PFX_FUNC_ALLOC)(size_t align, size_t size);
typedef void* (*SCE_PFX_FUNC_REALLOC)(void* ptr, size_t align, size_t size);
typedef void  (*SCE_PFX_FUNC_FREE)(void* ptr);
```

Serialization APIs

Structures

PfxSerializeCapacity

Structure storing size of physics instance subject to serialization processing

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
struct PfxSerializeCapacity
{
        PfxUInt32 maxRigidBodies;
        PfxUInt32 maxShapes;
        PfxUInt32 maxJoints;
        PfxUInt32 maxLargeMeshes;
        PfxUInt32 maxConvexMeshes;
        PfxUInt32 maxContacts;
        PfxUInt32 maxNonContactPairs;
        PfxUInt32 numRigidBodies;
        PfxUInt32 numShapes;
        PfxUInt32 numJoints;
        PfxUInt32 numLargeMeshes;
        PfxUInt32 numConvexMeshes;
        PfxUInt32 numContacts;
        PfxUInt32 numNonContactPairs;
};
```

Members

maxRigidBodies	Maximum number of rigid bodies
maxShapes	Maximum number of shapes
maxJoints	Maximum number of joints
maxLargeMeshes	Maximum number of large meshes
maxConvexMeshes	Maximum number of convex meshes
maxContacts	Maximum number of contacts
maxNonContactPairs	Maximum number of non-contact pairs
numRigidBodies	Number of rigid bodies
numShapes	Number of shapes
numJoints	Number of joints
numLargeMeshes	Number of large meshes
numConvexMeshes	Number of convex meshes
numContacts	Number of contacts
numNonContactPairs	Number of non-contact pairs

Description

For the serialization processing, this structure is used to notify to the serialization processing the maximum size and current size of the physics instance storage area being allocated at the application level.

In particular, when performing the serialization input processing, if the read data exceeds the maximum number set for this instance, the application must implement a callback function to reallocate the area.

PfxSerializeBuffer

Structure collecting the pointers to serialization target area

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
struct PfxSerializeBuffer
{
        PfxVector3* worldCenter;
        PfxVector3* worldExtent;
        PfxVector3* gravity;
        PfxRigidState* states;
        PfxRigidBody* bodies;
        PfxCollidable* collidables;
        PfxShape* shapes;
        PfxJoint* joints;
        PfxUInt32* nonContactPairs;
        PfxFloat* timeStep;
        PfxFloat* separateBias;
        PfxUInt32* iteration;
        PfxLargeTriMesh** largeMeshes;
        PfxConvexMesh** convexMeshes;
};
```

Members

worldCenter	Pointer to the world's center coordinates
worldExtent	Pointer to the world's extent
gravity	Pointer to the gravity
states	Pointer to the rigid body state buffer
bodies	Pointer to the rigid body attribute buffer
collidables	Pointer to the rigid body shape buffer
shapes	Pointer to the shape buffer
joints	Pointer to the joint buffer
nonContactPairs	Pointer to the non-contact pair buffer
timeStep	Pointer to the time step
separateBias	Pointer to the bias value
iteration	Pointer to the number of iterations
largeMeshes	Pointer to the large mesh buffer
convexMeshes	Pointer to the convex mesh buffer

Description

For the serialization processing, this structure stores the pointers to the physics instance storage area being allocated at the user application level.

The pointer destination buffer is used as the data storage destination during input, and as the output data reference source during output.

Note that the large mesh buffer and the convex mesh buffer constitute the pointer array for each mesh, respectively. During input, allocate the memory for meshes internally, and store the pointer to the array. This pointer is set to the rigid body shape that refers to the mesh. Implement release of the mesh data at the application level. If using a user-specified memory allocator for the memory allocation for mesh data, make this setting using `pfxSetUtilityAllocator()`.

Functions

pfxSerializeRead

Serialization processing for inputting data from a file

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
PfxInt32 pfxSerializeRead (
        void* fh,
        PfxSerializeInitFunc funcInit,
        PfxSerializeTermFunc funcTerm,
        PfxSerializeUpdateFunc funcUpdate,
        PfxSerializeResizeFunc funcResize,
        PfxSerializeErrorFunc funcError = 0,
        eFileFormat format = kSnapshot
)
```

Arguments

fh	`FILE` type pointer of the C standard library
funcInit	Pointer to the initialization callback function for serialization
funcTerm	Pointer to the termination callback function for serialization
funcUpdate	Pointer to the update callback function for serialization
funcResize	Pointer to the buffer resizing callback function at serialization
funcError	Pointer to the error handling callback function that is called when an error arises
format	Input format (currently only `kSnapshot` is valid)

Return Values

Returns `SCE_PFX_OK(0)` for normal termination.

Returns the error `SCE_PFX_ERR_INVALID_VALUE` if the parameter is invalid.

Description

This function is used to read a file that has been serialized beforehand. The read data is stored in the buffer specified in initialization callback function. The user must open an input file through the C standard library (text mode) and prepare *fh*, a pointer to the `FILE` type instance beforehand. The file will not be closed within the serialization processing. Thus, the application must close the file after the serialization has ended.

Each callback function specified to the arguments is called at the appropriate timing during the serialization processing. The initialization callback function is called prior to serialization, and the destination buffer and size information are set to structures `PfxSerializeCapacity` and `PfxSerializeBuffer` and are passed to serialization processing.

The amount of data stored in the file and the specified buffer capacity are compared, and if the buffer size is insufficient, the buffer resizing callback function is called. Perform buffer reallocation in the buffer resizing callback function and return `true`. If `false` is returned, or if the buffer resizing callback function has not been specified, capacity allocation is considered to have failed, the serialization processing is aborted, and the error handling callback function is called.

Upon completion of loading, after the data has been stored to the buffer, the termination callback function is called. Describe additional setups and other processing to this callback function.

During file reading, the update callback function can be called periodically to check the progress status.

In case of an error, the error handling callback function will be called.

pfxSerializeWrite

Serialization processing for outputting data into a file

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
PfxInt32 pfxSerializeWrite (
        void* fh,
        PfxSerializeInitFunc funcInit,
        PfxSerializeTermFunc funcTerm,
        PfxSerializeErrorFunc funcError = 0,
        eFileFormat format = kSnapshot
)
```

Arguments

fh	`FILE` type pointer of the C standard library
funcInit	Pointer to the initialization callback function for serialization
funcTerm	Pointer to the termination callback function for serialization
funcError	Pointer to the error handling callback function that is called when an error arises
format	Input format (currently only `kSnapshot` is valid)

Return Values

Returns `SCE_PFX_OK(0)` for normal termination of serialization processing

Returns the error `SCE_PFX_ERR_INVALID_VALUE` if the parameter is invalid.

Description

This function is used to write the Physics Effects instance into a file.

The user must open an input file through the C standard library and prepare a pointer to the `FILE` type instance beforehand. Be sure to open the file in writable binary mode. The file will not be closed within the serialization processing. Thus, the user must close the file after the serialization has ended.

Each callback function specified to the arguments is called at the appropriate timing during the serialization processing. The initialization callback function is called prior to serialization, and the write out reference source buffer and size information are set to structures `PfxSerializeCapacity` and `PfxSerializeBuffer` and are passed to serialization processing.

Upon completion of the write processing, the termination callback function is called. If necessary, execute the postprocessing within this function.

In case of an error, the error handling callback function will be called.

Callback Functions

PfxSerializeInitFunc

Initialization callback

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
typedef void(*PfxSerializeInitFunc)(
        PfxSerializeCapacity *capacity,
        PfxSerializeBuffer *buffer
);
```

Arguments

capacity	Structure storing sizes of physics instances
buffer	Structure storing the pointers to the buffers

Return Values

None

Description

This callback function is called during serialization initialization.

If it is called from `pfxSerializeRead()`, the size of each physics instance is set to *capacity*, and the information of the pointer to the storage destination buffer is set to *buffer*. In the case of insufficient capacity, the resizing callback function is called.

If this function is called from `pfxSerializeWrite()`, the size of each physics instance is set to *capacity*, and the information of the pointer to the reference source buffer is set to *buffer*.

PfxSerializeTermFunc

Termination callback

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
typedef void(*PfxSerializeTermFunc)(
        PfxSerializeCapacity *capacity,
        PfxSerializeBuffer *buffer
);
```

Arguments

capacity	Structure storing sizes of physics instances
buffer	Structure storing the pointers to the buffers

Return Values

None

Description

This callback function is called upon completion of the serialization processing.

If it is called from `pfxSerializeRead()`, the size of the physics instance that is actually loaded is stored in *capacity*, and the data is stored in the buffers indicated by the pointers in *buffer*.

PfxSerializeResizeFunc

Buffer resizing callback

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
typedef PfxBool(*PfxSerializeResizeFunc)(
        PfxSerializeCapacity *capacity,
        PfxSerializeBuffer *buffer,
        const PfxSerializeCapacity *capacityLoaded
);
```

Arguments

capacity	Structure storing sizes of physics instances
buffer	Structure storing the pointers to the buffers
capacityLoaded	Structure storing the sizes of physics instances to be loaded

Return Values

Returns `true` when the buffer reallocation was successful, and `false` when it failed.

Description

This function is called when the capacity of the buffer specified with the initialization callback function during file loading is insufficient. Because the capacity that is actually required is set to *capacityLoaded*, compare that value with the size specified to *capacity*, and reallocate the buffer for size of insufficiency.

PfxSerializeUpdateFunc

Update callback

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
typedef void(*PfxSerializeUpdateFunc)(
        PfxUInt32 progress,
        PfxUInt32 maxProgress
);
```

Arguments

progress	Position during progress
maxProgress	Position at completion

Return Values

None

Description

This function, which is called periodically during serialization input, reports the current progress status to the application.

PfxSerializeErrorFunc

Error handling callback

Definition

```
#include <physics_effects/util/pfx_serialize_ex.h>
typedef void(*PfxSerializeErrorFunc)(
        PfxInt32 errorCode,
        const char **tags,
        int numTags
);
```

Arguments

errorCode	Error code
tags	Array of tag character strings
numTags	Number of tags stored to array

Return Values

None

Description

This function is called when an error occurs during serialization input/output. When an error occurs during file analysis, the tag information for determining the occurrence location is set along with the error code. For the tag information, the hierarchy from the root in the file is returned as an array of tag character strings.

One of the following is set for the error code.

Value	Description
`SCE_PFX_ERR_SERIALIZE_INVALID_FILE`	File loading failed
`SCE_PFX_ERR_SERIALIZE_INVALID_VERSION`	The format version is different
`SCE_PFX_ERR_SERIALIZE_INVALID_TAG`	An unidentifiable tag was found
`SCE_PFX_ERR_SERIALIZE_INVALID_FORM`	The data file format contains an error
`SCE_PFX_ERR_SERIALIZE_INVALID_COUNT`	Data inconsistency occurred

Debug Rendering APIs

Structures

PfxLargeMeshFlagTable

Structure managing the table of debug rendering type flag of large mesh

Definition

```
#include <physics_effects/base_level/sort/pfx_sort_data.h>
Struct PfxLargeMeshFlagTable {
        PfxUInt32 rigidbodyId;
        PfxUInt8  * largeMeshFlagTable;
};
```

Members

rigidbodyId	Index of rigid body
largeMeshFlagTable	Pointer to the flag table that manages the type of large mesh debug rendering

Description

This structure manages both the rigid body index and the table storing the debug rendering type flag of each triangle to which a large mesh is set.

Classes

PfxDebugRender

Debug rendering class

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {};
```

Description

This class provides functions to visualize the information in the physics engine for the debugging purpose. At the application level, functions to execute basic rendering processing to render points, lines, boxes, etc., are provided so as to prevent dependence on specific rendering APIs, and they are called through the user callback functions when debug rendering is executed. Before calling the debug rendering functions, it is necessary to set the required basic rendering processing function for each debug rendering function.

Method List

Method	Description
`setDebugRenderPointFunc`	Sets the point rendering function
`setDebugRenderLineFunc`	Sets the line rendering function
`setDebugRenderArcFunc`	Sets the arc rendering function
`setDebugRenderAabbFunc`	Sets the bounding box rendering function
`setDebugRenderBoxFunc`	Sets the box rendering function
`resetVisible`	Resets the debug rendering visibility flag of a rigid body
`getVisible`	Gets the debug rendering visibility flag of each rigid body
`setVisible`	Sets the debug rendering visibility flag of each rigid body
`renderWorld`	Renders the bounding box in the world area
`renderAabb`	Renders the bounding box of the rigid body
`renderLocalAxis`	Renders the local coordinate axes
`renderIsland`	Renders the simulation island
`renderContact`	Renders the collision information
`renderLargeMesh`	Renders the debug information of a large mesh
`renderLargeMeshInFrustum`	Renders the debug information of the large meshes in the view frustum
`renderLargeMeshByFlag`	Renders the debug information of a large mesh according to the flag table
`renderJoint`	Renders the debug information of a joint
`resetLargeMeshFlagTables`	Resets the rendering type flag table of a large mesh
`getLargeMeshFlag`	Gets the rendering type flag of a large mesh
`setLargeMeshFlag`	Sets the rendering type flag of a large mesh
`getScale`	Gets the rendering scale of debug information
`setScale`	Sets the rendering scale of debug information

Constructors and Destructors

PfxDebugRender

Constructor of debug rendering class

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender{
        PfxDebugRender();
};
```

Arguments

None

Return Values

None

Description

This constructor is used to initialize the debug rendering class.

~PfxDebugRender

Destructor of debug rendering class

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        ~PfxDebugRender();
};
```

Arguments

None

Return Values

None

Description

This destructor is used to clear the internal buffer.

Public Methods

setDebugRenderPointFunc

Set point rendering function

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setDebugRenderPointFunc(PfxDebugRenderPointFunc func);
};
```

Arguments

func	Function pointer to the point rendering function

Return Values

None

Description

This public method sets the point rendering function prepared at the application level.

The point rendering function must be set through this method before `renderContact()` and `renderJoint()` are used. Assert is called if the function is not set.

setDebugRenderLineFunc

Set line rendering function

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setDebugRenderLineFunc(PfxDebugRenderLineFunc func);
};
```

Arguments

func	Function pointer to the line rendering function

Return Values

None

Description

This public method sets the line rendering function prepared at the application level.

The line rendering function must be set through this method before `renderContact()`, `renderLocalAxis()`, `renderLargeMeshByFlag()`, `renderLargeMesh()` and `renderJoint()` are used. Assert is called if the function is not set.

setDebugRenderArcFunc

Set arc rendering function

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setDebugRenderArcFunc(PfxDebugRenderArcFunc func);
};
```

Arguments

func	Function pointer to the arc rendering function

Return Values

None

Description

This public method sets the arc rendering function prepared at the application level.

The arc rendering function must be set through this method before `renderJoint()` is used. Assert is called if the function is not set.

setDebugRenderAabbFunc

Set bounding box rendering function

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setDebugRenderAabbFunc(PfxDebugRenderAabbFunc func);
};
```

Arguments

func	Function pointer to the bounding box rendering function

Return Values

None

Description

This public method sets the bounding box rendering function prepared at the application level.

The bounding box rendering function must be set through this method before `renderWorld()`, `renderAabb()` and `renderIsland()` are used. Assert is called if the function is not set.

setDebugRenderBoxFunc

Set box rendering function

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setDebugRenderBoxFunc(PfxDebugRenderBoxFunc func);
};
```

Arguments

func	Function pointer to the box rendering function

Return Values

None

Description

This public method sets the box rendering function prepared at the application level.

The box rendering function must be set through this method before `renderContact()`, `renderLocalAxis()`, `renderLargeMeshByFlag()`, `renderLargeMesh()` and `renderJoint()` are used. Assert is called if the function is not set.

resetVisible

Reset debug rendering visibility flag of rigid body

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void resetVisible (
              const PfxUInt32 numRigidbodies
        );
};
```

Arguments

numRigidbodies	Number of rigid bodies

Return Values

None

Description

This public method discards the debug rendering visibility flag of the rigid bodies being held and rebuilds the visibility flag from the number of rigid bodies given as an argument. Flags in the rebuilt table are set with "Show".

getVisible

Get debug rendering visibility flag of each rigid body

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        PfxInt32 getVisible (
              const PfxUInt32 rigidbodyId,
              PfxUInt8 &flag
        );
};
```

Arguments

rigidbodyId	Index of rigid body to be obtained
flag	Debug rendering visibility flag of rigid body to be obtained

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is invalid and error `SCE_PFX_ERR_INVALID_FLAG` if the flag value is invalid.

Description

This public method gets the rendering type flag set through the `setVisible()` method.

The following values are set to *flag*.

Value	Description
`SCE_PFX_DRENDER_INVISIBLE`	Hide
`SCE_PFX_DRENDER_VISIBLE`	Show

setVisible

Set debug rendering visibility flag of each rigid body

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        PfxInt32 setVisible (
              const PfxUInt32 rigidbodyId,
              const PfxUInt8 flag
        );
};
```

Arguments

rigidbodyId	Index of rigid body to be set
flag	Debug rendering visibility flag of rigid body to be set

Return Values

Returns `SCE_PFX_OK(0)` upon normal termination.

Returns the error `SCE_PFX_ERR_OUT_OF_RANGE` if the parameter range is invalid and error `SCE_PFX_ERR_INVALID_FLAG` if the flag value is invalid.

Description

This public method sets the debug rendering visibility flag to the rigid body specified with *rigidbodyId*.

The following values are set to *flag*.

Value	Description
`SCE_PFX_DRENDER_INVISIBLE`	Hide
`SCE_PFX_DRENDER_VISIBLE`	Show

renderWorld

Render bounding box in world area

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderWorld(
             const PfxVector3 &center,
             const PfxVector3 &halfExtent
        );
};
```

Arguments

center	Center coordinates value of bounding box in the world area
halfExtent	Size of bounding box in the world area

Return Values

None

Description

This public method renders the bounding box in the world area.

The bounding box rendering function must be set through `setDebugRenderAabbFunc()` beforehand.

Assert is called if the function is not set.

renderAabb

Render bounding box of rigid body

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderAabb(
              const PfxRigidState *states,
              const PfxCollidable *collidables,
              const PfxUInt32 numRigidbodies
        );
};
```

Arguments

states	Rigid body state
collidables	Rigid body shape
numRigidbodies	Number of rigid bodies

Return Values

None

Description

This public method renders the bounding box of each rigid body.

The bounding box rendering function must be set through `setDebugRenderAabbFunc()` beforehand.

Assert is called if the function is not set.

renderLocalAxis

Render local coordinate axes

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderLocalAxis(
             const PfxRigidState *states,
             const PfxUInt32 numRigidbodies
        );
};
```

Arguments

states	Rigid body state
numRigidbodies	Number of rigid bodies

Return Values

None

Description

This public method renders the local coordinate axes of each rigid body.

The line rendering function must be set through `setDebugRenderLineFunc()` beforehand.

Assert is called if the function is not set.

renderIsland

Render simulation island

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderIsland(
              const PfxIsland *island,
              const PfxRigidState *states,
              const PfxCollidable *collidables
        );
};
```

Arguments

island	Simulation Island
states	Rigid body state
collidables	Rigid body shape

Return Values

None

Description

This public method renders the simulation island.

The bounding box rendering function must be set through `setDebugRenderAabbFunc()` beforehand.

Assert is called if the function is not set.

renderContact

Render collision information

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderContact(
             const PfxContactManifold *contacts,
             const PfxBroadphasePair *pairsBuff,
             const PfxRigidState *states,
             const PfxUInt32 numContacts
        );
};
```

Arguments

contacts	Contact information
pairsBuff	Pair buffer
states	Rigid body state
numContacts	Number of contacts

Return Values

None

Description

This public method renders each collision point and normal vector.

The point rendering function and line rendering function must be set through `setDebugRenderPointFunc()` and `setDebugRenderLineFunc()` respectively in advance.

Assert is called if the function is not set.

renderLargeMesh

Render debug information of large mesh

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderLargeMesh(
              const PfxRigidState *states,
              const PfxCollidable *collidables,
              const PfxUInt32 numRigidbodies,
              const PfxUInt32 flag
        );
};
```

Arguments

states	Rigid body state
collidables	Rigid body shape
numRigidbodies	Number of rigid bodies
flag	Mesh rendering type flag

Return Values

None

Description

This public method performs rendering processing for all the flags to which the island mesh, edge of the triangle and AABB of the triangle, which are collectively referred to as the debug information of each large mesh, are specified.

The line rendering function and box rendering function must be set through `setDebugRenderLineFunc()` and `setDebugRenderBoxFunc()` respectively in advance.

Assert is called if the function is not set.

renderLargeMeshInFrustum

Render debug information of large mesh in view frustum

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderLargeMeshInFrustum (
              const PfxRigidState *states,
              const PfxCollidable *collidables,
              const PfxUInt32 numRigidbodies,
              const PfxUInt32 flag,
              const PfxVector4* planes,
              int numPlanes
        );
};
```

Arguments

states	Rigid body state
collidables	Rigid body shape
numRigidbodies	Number of rigid bodies
flag	Mesh rendering type flag
planes	Array of planes making up view frustum
numPlanes	Number of planes making up view frustum

Return Values

None

Description

This public method renders all the debug information on the islands included in the specified view frustum for the rendering flags to which the island mesh, edge of the triangle and AABB of the triangle, which are collectively referred to as the debug information of each large mesh, are specified.

The line rendering function and box rendering function must be set through `setDebugRenderLineFunc()` and `setDebugRenderBoxFunc()`, respectively, in advance.

Assert is called if the functions are not set.

The range that is displayed depends on the value of *planes*.

numPlanes value	Description
0	Renders everything (same as `renderLargeMesh()`)
1	Renders the triangles included in the island whose center coordinate and radius consist of the first three elements and the fourth element, respectively, of *planes*[0].
2 or higher	With the plane whose normal vector consists of the first three elements and whose distance from the origin consists of the fourth element, respectively, of each *planes*, as the boundary, renders the triangles included in the island whose normal vector points in the opposite direction.

renderLargeMeshByFlag

Render debug information of large mesh according to the flag table

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderLargeMeshByFlag(
              const PfxRigidState *states,
              const PfxCollidable *collidables,
              const PfxUInt32 flag
        );
};
```

Arguments

states	Rigid body state
collidables	Rigid body shape
flag	Mesh rendering type flag

Return Values

None

Description

Among the mesh rendering type flag table set through `setLargeMeshFlag()` in advance, this public method only renders the debug information of the large mesh specified with *flag*.

The line rendering function and box rendering function must be set through `setDebugRenderLineFunc()` and `setDebugRenderBoxFunc()` respectively in advance.

Assert is called if the function is not set.

renderJoint

Render debug information of joint

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void renderJoint(
              const PfxJoint *joints,
              const PfxRigidState *states,
              const PfxUInt32 numJoints
        );
};
```

Arguments

joints	Joint structure
states	Rigid body state
numJoints	Number of joints

Return Values

None

Description

This public method renders the debug information of each joint.

The point, line and arc rendering functions must be set through `setDebugRenderPointFunc()`, `setDebugRenderLineFunc()` and `setDebugRenderArcFunc()` respectively in advance.

Assert is called if the function is not set.

resetLargeMeshFlagTables

Reset rendering type flag table of large mesh

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void resetLargeMeshFlagTables(
              const PfxCollidable *collidables,
              const PfxUInt32 numRigidbodies
        );
};
```

Arguments

collidables	Rigid body shape
numRigidbodies	Number of rigid bodies

Return Values

None

Description

This public method discards the rendering type flag table of the mesh being held and rebuilds the new rendering type flag table from the rigid body shape given as an argument. Flags in the rebuilt table are cleared with 0.

Assert is called if the rebuilding of the flag table fails.

getLargeMeshFlag

Get rendering type flag of large mesh

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        PfxInt32 getLargeMeshFlag(
              const PfxUInt32 rigidbodyId,
              const PfxUInt32 islandId,
              const PfxUInt32 facetId,
              PfxUInt8 &flag
        );
};
```

Arguments

rigidbodyId	Index of the rigid body to which the mesh is set
islandId	Index of the island mesh
facetId	Index of the triangle
flag	Rendering type flag to be acquired

Return Values

Returns `SCE_PFX_OK(0)` for normal termination.

Returns an error of `SCE_PFX_ERR_OUT_OF_RANGE` if the range of parameter is invalid.

Description

This public method gets the rendering type flag set through the `setLargeMeshFlag()` method.

Combination of the following values is set to *flag*.

Value	Description
SCE_PFX_DRENDER_MESH_FLG_NONE	No flag is set
SCE_PFX_DRENDER_MESH_FLG_ISLAND	Renders the specified island mesh
SCE_PFX_DRENDER_MESH_FLG_EDGE	Renders the specified edge of triangle
SCE_PFX_DRENDER_MESH_FLG_FACET_AABB	Renders the specified bounding box (AABB) of triangle
SCE_PFX_DRENDER_MESH_FLG_NORMAL	Renders the normal of triangle
SCE_PFX_DRENDER_MESH_FLG_THICKNESS	Renders the thickness of triangle
SCE_PFX_DRENDER_MESH_FLG_ALL	Specifies all the flags

setLargeMeshFlag

Set rendering type flag of large mesh

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        PfxInt32 setLargeMeshFlag(
              const PfxUInt32 rigidbodyId,
              const PfxUInt32 islandId,
              const PfxUInt32 facetId,
              const PfxUInt8 flag
        );
};
```

Arguments

rigidbodyId	Index of the rigid body to which the mesh is set
islandId	Index of the island mesh
facetId	Index of the triangle
flag	Rendering type flag

Return Values

Returns `SCE_PFX_OK(0)` for normal termination.

Returns an error of `SCE_PFX_ERR_OUT_OF_RANGE` if the range of parameter is invalid.

Description

This public method sets a flag to the rendering type flag table of the large mesh specified with *rigidbodyId*.

to *flag* specify combination of the following values.

Value	Description
SCE_PFX_DRENDER_MESH_FLG_NONE	Clears the flag
SCE_PFX_DRENDER_MESH_FLG_ISLAND	Renders the island mesh
SCE_PFX_DRENDER_MESH_FLG_EDGE	Renders the edge of triangle
SCE_PFX_DRENDER_MESH_FLG_FACET_AABB	Renders the bounding box (AABB) of triangle
SCE_PFX_DRENDER_MESH_FLG_NORMAL	Renders the normal of triangle
SCE_PFX_DRENDER_MESH_FLG_THICKNESS	Renders the thickness of triangle
SCE_PFX_DRENDER_MESH_FLG_ALL	Specifies all the flags

getScale

Get rendering scale of debug information

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        PfxFloat getScale();
};
```

Arguments

None

Return Values

Returns the value of the scale

Description

This public method returns the rendering scale value of the debug information.

setScale

Set rendering scale of debug information

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
class PfxDebugRender {
        void setScale(PfxFloat scale);
};
```

Arguments

scale	Scale

Return Values

None

Description

This public method sets the rendering scale value of the debug information.

Callback Functions

PfxDebugRenderPointFunc

Callback function that is called during point rendering

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
typedef void (*PfxDebugRenderPointFunc)(
        const PfxVector3 &position,
        const PfxVector3 &color
);
```

Arguments

position	Position of point
color	Color of point

Return Values

None

Description

Implement the codes for rendering points at the specified positions.

PfxDebugRenderLineFunc

Callback function that is called during line rendering

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
typedef void (*PfxDebugRenderLineFunc)(
        const PfxVector3 &position1,
        const PfxVector3 &position2,
        const PfxVector3 &color
);
```

Arguments

position1	Start position of the line
position2	End position of the line
color	Color of the line

Return Values

None

Description

Implement the codes for rendering lines at the specified positions.

PfxDebugRenderArcFunc

Callback function that is called during arc rendering

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
typedef void (*PfxDebugRenderArcFunc)(
        const PfxVector3 &pos,
        const PfxVector3 &axis,
        const PfxVector3 &dir,
        const float radius,
        const float startRad,
        const float endRad,
        const PfxVector3 &color
);
```

Arguments

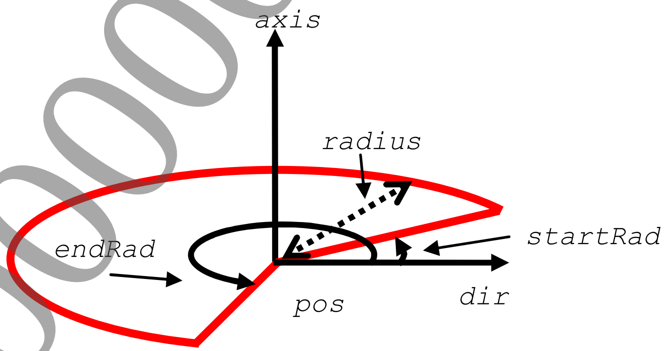
pos	Center of the arc
axis	Axis of rotation of the arc
dir	Vector from the center to the arc
radius	Radius of the arc
startRad	Rendering start angle of the arc
endRad	Rendering end angle of the arc
color	Color of the arc

Return Values

None

Description

Implement the codes for rendering arcs.



PfxDebugRenderAabbFunc

Callback function that is called during bounding box rendering

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
typedef void (*PfxDebugRenderAabbFunc)(
        const PfxVector3 &center,
        const PfxVector3 &halfExtent,
        const PfxVector3 &color
);
```

Arguments

center	Center of the bounding box
halfExtent	Extent of the bounding box
color	Color of the bounding box

Return Values

None

Description

Implement the codes for rendering boxes parallel to the XYZ axes.

PfxDebugRenderBoxFunc

Callback function that is called during box rendering

Definition

```
#include <physics_effects/util/pfx_debug_render.h>
typedef void (*PfxDebugRenderBoxFunc)(
        const PfxTransform3 &transform,
        const PfxVector3 &halfExtent,
        const PfxVector3 &color
);
```

Arguments

transform	Transform of the box
halfExtent	Extent of the box
color	Color of the box

Return Values

None

Description

Implement the codes for rendering boxes.