Overlap Testing

In addition to providing Raycast and Trigger support, the NVIDIA PhysX SDK provides a set of general methods for testing collision primitive directly against shapes. These are useful when the other collision methods do not fit the user's needs.

The methods provided by NxScene allow efficient testing against the entire collision database, while the methods of NxShape test against specific objects.

NxTriangleMesh also provides methods for testing an AABB against the triangle mesh and retrieving the set of triangles which overlap the AABB.

NOTE: Because the SDK double buffers data to allow asynchronous stepping state modification is not visible to some functions(eg overlap and raycasting) until a simulate()/fetchResults() pair has been executed.

NxScene::overlapSphereShapes()

```
NxU32 overlapSphereShapes(const NxSphere& worldSphere, NxShapesType shapeType, NxU32 nbShapes, NxShape** shapes,
NxUserEntityReport<NxShape*>* callback, NxU32 activeGroups=0xffffffff, const NxGroupsMask* groupsMask=NULL,
bool accurateCollision=false);
```

This method tests a sphere (in the global frame) against shapes in the scene. The user can test against static, dynamic, or all shapes by setting the shapeType parameter to NX_STATIC_SHAPES, NX_DYNAMIC_SHAPES or NX_ALL_SHAPES respectively.

The user has two options for retrieving the overlapping shapes, detailed below:

- Provide a buffer large enough to hold all the overlapping shapes using the shapes and nbShapes parameters. If this buffer is insufficient in size, then only the first nbShapes are returned.
- Provide an NxUserEntityReport callback which directs the SDK to call the callback with sets of shapes as they are detected. A small buffer is allocated on the stack to return shapes, unless a buffer is provided using the shapes parameter, in which case the SDK uses this temporarily.

The activeGroups and groupsMask parameters provide an optional mechanism for filtering shapes based on their group membership. See Broad Phase Collision Detection for more information.

The accurateCollision parameter provides the option to either collide against the actual shapes (true) or the AABBs (false) shapes.

Example

```
NxSphere worldSphere(NxVec3(0,0,0), 5);

NxU32 nbShapes = gScene->getNbDynamicShapes();

NxShape** shapes = new NxShape* [nbShapes];

for (NxU32 i = 0; i < nbShapes; i++)
    shapes[i] = NULL;

gScene->overlapSphereShapes(worldSphere, NX_DYNAMIC_SHAPES, nbShapes, shapes, NULL);
```

NxScene::overlapAABBShapes()

```
NxU32 overlapAABBShapes(const NxBounds3& worldBounds, NxShapesType shapeType, NxU32 nbShapes, NxShape** shapes,
NxUserEntityReport<NxShape*>* callback, NxU32 activeGroups=0xffffffff, const NxGroupsMask* groupsMask=NULL,
bool accurateCollision=false);
```

This method is identical to overlapSphereShapes(), except it tests an axis aligned bound box against the shape database instead of a sphere.

Example

```
NxBounds3 worldBounds;
worldBounds.set(NxVec3(-5,-5,-5), NxVec3(5,5,5));

NxU32 nbShapes = gScene->getNbDynamicShapes();

NxShape** shapes = new NxShape* [nbShapes];

for (NxU32 i = 0; i < nbShapes; i++)
    shapes[i] = NULL;

gScene->overlapAABBShapes(worldBounds, NX_DYNAMIC_SHAPES, nbShapes, shapes, NULL);
```

NxScene::overlapOBBShapes()

```
NxU32 overlapOBBShapes(const NxBox& worldBox, NxShapesType shapeType, NxU32 nbShapes, NxShape** shapes,
NxU\serEntityReport<NxShape*>* callback, NxU32 activeGroups=0xffffffff, const NxGroupsMask* groupsMask=NULL,
bool accurateCollision=false) = 0;
```

This method is identical to overlapSphereShapes(), except it tests an oriented bounding box against the shape database instead of a sphere.

NxScene::overlapCapsuleShapes()

```
NxU32 overlapCapsuleShapes(const NxCapsule& worldCapsule, NxShapesType shapeType, NxU32 nbShapes, NxShape** shapes,
NxUserEntityReport<NxShape*>* callback, NxU32 activeGroups=0xffffffff, const NxGroupsMask* groupsMask=NULL,
bool accurateCollision=false) = 0;
```

This method is identical to overlapSphereShapes(), except it tests a world space capsule against the shape database instead of a sphere.

NxScene::cullShapes()

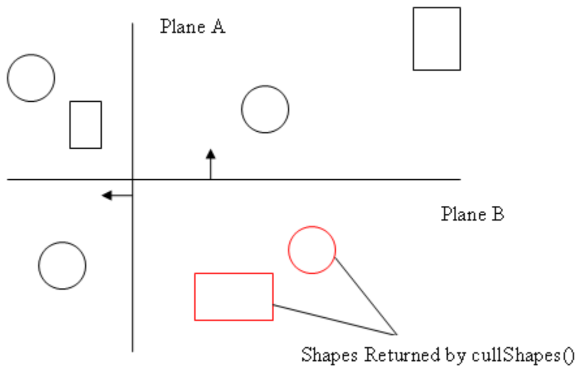
```
NxU32 cullShapes(NxU32 nbPlanes, const NxPlane* worldPlanes, NxShapesType shapeType, NxU32 nbShapes, NxShape** shapes,
NxUserEntityReport<NxShape*>* callback, NxU32 activeGroups=0xffffffff, const NxGroupsMask* groupsMask=NULL);
```

This method is a little different; it tests all shapes in the collision database against a set of planes. It tests objects against the planes and returns them if they are in the plane's negative half space (i.e., on the side which the normal points away from).

Note that the set of shapes returned by cullShapes() is not conservative; additional shapes may be returned which do not intersect the volume formed by the plane's positive half space. This allows a more efficient implementation.

A limitation of this function is that the number of planes should not exceed 32.

The primary intent of this function is for view frustum culling. The user can pass through the 6 planes which define the view frustum to receive all the objects which intersect it for further processing.



Example

```
NxPlane worldPlanes[2];

worldPlanes[0]=NxPlane(-1.0f,0.0f,0.0f,0.0f);
worldPlanes[1]=NxPlane(0.0f,1.0f,0.0f,0.0f);

NxU32 nbShapes = gScene->getNbDynamicShapes();

NxShape** shapes = new NxShape* [nbShapes];

for (NxU32 i = 0; i < nbShapes; i++)
    shapes[i] = NULL;

gScene->cullShapes(2, worldPlanes, NX_DYNAMIC_SHAPES, nbShapes, shapes, NULL);
```

NxScene::checkOverlapSphere()

```
bool checkOverlapSphere(const NxSphere& worldSphere, NxShapesType shapeType=NX_ALL_SHAPES, NxU32 activeGroups=0xffffffff,
    const NxGroupsMask* groupsMask=NULL, bool accurateCollision=false);
```

This method tests the sphere against shapes in the collision database. The "accurateCollision" parameter controls if this method should return true only if the sphere overlaps the actual shape or the AABB of the shape (accurateCollision=false). The default is to only test against the AABB of the shape.

Example

```
NxSphere worldSphere(NxVec3(0,0,0), 5);

bool intersection = gScene->checkOverlapSphere(worldSphere, NX_DYNAMIC_SHAPES);
```

NxScene::checkOverlapAABB()

```
bool checkOverlapAABB(const NxBounds3& worldBounds, NxShapesType shapeType=NX_ALL_SHAPES, NxU32 activeGroups=0xffffffff,
const NxGroupsMask* groupsMask=NULL, bool accurateCollision=false);
```

This method tests the specified AABB against shapes in the collision database. The "accurateCollision" parameter controls if this method should return true only if the specified AABB overlaps the actual shape or the AABB of the shape (accurateCollision=false). The default is to only test against the AABB of the shape.

Example

```
NxBounds3 worldBounds;
worldBounds.set(NxVec3(-5,-5,-5), NxVec3(5,5,5));

bool intersection = gScene->checkOverlapAABB(worldBounds, NX_DYNAMIC_SHAPES);
```

NxScene::checkOverlapCapsule()

```
bool checkOverlapCapsule(const NxCapsule& worldCapsule, NxShapesType shapeType=NX_ALL_SHAPES, NxU32 activeGroups=0xffffffff,
    const NxGroupsMask* groupsMask=NULL, bool accurateCollision=false);
```

This method tests the capsule against shapes in the collision database. The "accurateCollision" parameter controls if this method should return true only if the capsule overlaps the actual shape or the AABB of the shape (accurateCollision=false). The default is to only test against the AABB of the shape.

Example

```
NxCapsule worldCapsule(NxSegment(NxVec3(0.0f,0.0f,0.0f),NxVec3(0.0f,1.0f,0.0f)),1.0f);

bool intersection = gScene->checkOverlapCapsule(worldCapsule, NX_DYNAMIC_SHAPES);
```

NxScene::checkOverlapOBB()

```
bool checkOverlapOBB(const NxBox& worldBox, NxShapesType shapeType=NX_ALL_SHAPES, NxU32 activeGroups=0xffffffff,
    const NxGroupsMask* groupsMask=NULL, bool accurateCollision=false);
```

This method tests the OBB against shapes in the collision database. The "accurateCollision" parameter controls if this method should return true only if the OBB overlaps the actual shape or the AABB of the shape (accurateCollision=false). The default is to only test against the AABB of the shape.

Example

```
NxBox worldBox(NxVec3(0.0f,0.0f,0.0f),NxVec3(1.0f,1.0f,1.0f), NxMat33(NX_IDENTITY_MATRIX));

bool intersection = gScene->checkOverlapOBB(worldBox, NX_DYNAMIC_SHAPES);
```

NxShape::checkOverlapSphere()

```
bool checkOverlapSphere(const NxSphere& worldSphere);
```

This method is similar to NxScene::checkOverlapSphere(), except that only a single shape is tested.

NxShape::checkOverlapAABB()

```
bool checkOverlapAABB(const NxBounds3& worldBounds);
```

This method is similar to NxScene::checkOverlapAABB(), except that only a single shape is tested.

NxShape::checkOverlapCapsule()

```
bool checkOverlapCapsule(const NxCapsule& worldCapsule);
```

This method is similar to NxScene::checkOverlapCapsule(), except that only a single shape is tested.

NxShape::checkOverlapOBB()

```
bool checkOverlapOBB(const NxBox& worldBox);
```

This method is similar to NxScene::checkOverlapOBB(), except that only a single shape is tested.

NxTriangleMeshShape::overlapAABBTriangles() and NxHeightFieldShape::overlapAABBTriangles()

```
bool overlapAABBTriangles(const NxBounds3& bounds, NxU32 flags, NxUserEntityReport<NxU32>* callback);
```

Use this method to test an AABB against a triangle mesh shape and retrieve a list of triangles which intersect the AABB. The user entity report(callback) is called with batches of triangle indices.

Triangle indices returned by overlapAABBTriangles() can be used with getTriangle() to obtain the triangle's position, edge normals and flags.

The flag's member provides the following options:

- NX_QUERY_WORLD_SPACE - If set, the AABB is specified in the global frame. Otherwise, it is assumed to be specified in the shape's frame.
- NX_QUERY_FIRST_CONTACT - If set, the overlapAABBTriangles() only returns the first intersecting triangle that it detects.

NxTriangleMeshShape::getTriangle()

```
NxU32 getTriangle(NxTriangle& worldTri, NxTriangle* edgeTri, NxU32* flags,
            NxTriangleID triangleIndex, bool worldSpaceTranslation=true, bool worldSpaceRotation=true);
```

This method allows the user to retrieve information about a triangle specified by its index (triangleIndex). The edgeTri and flags parameters are optional and can be set to NULL.

- worldTri - The world space vertex positions of the triangle.
- edgeTri - The world space edge normals of the triangle.
- flags - Specify if an edge of the triangle is convex or not. NXTF_ACTIVE_EDGE01, NXTF_ACTIVE_EDGE12, NXTF_ACTIVE_EDGE20 are set if edge 0, 1 or 2 are convex.
- worldSpaceRotation, worldSpaceTranslation - allows you to get the result in the world frame or in the local frame, with any combination of world or local rotation/translation.

Caveat

When shapes' poses are updated by the application directly, such as through calling NxActor::setGlobalPose, their broad phase bounds are not automatically updated until the next simulate call. This may cause incorrect results for API calls like overlap tests or cloth attachment.
A newly created entity has as yet no bounds and is not affected by this shortcoming.
If you are experiencing problems with for example overlap testing because of this, you can work around it by deferring such relocations until after tests are performed.

API Reference

- NxScene
- NxShape
- NxTriangleMeshShape
