PhysX migration

(2.8.1 -> 3.3.4)

2.8.1

3.3.4

API Interface : Actors - Dynamic

2.8.1

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;

NxBoxShapeDesc boxDesc;
boxDesc.dimensions              = NxVec3(1.5f, 1.5f, 1.5f);
actorDesc.shapes.pushBack(&boxDesc);

actorDesc.density               = 1.0f;
actorDesc.body                  = &bodyDesc;
actorDesc.globalPose.t          = NxVec3(3.0f, 0.0f, 0.0f);

NxActor *pActor = gScene->createActor(actorDesc);
```

Description → Actor

3.3.4

```
PxRigidDynamic* dynamic =
                    PxCreateDynamic(sdk, trans, geom, *material, 1.0f);
```

```
PxRigidDynamic* dynamic =
                    sdk>createRigidDynamic(trans);

PxShape* box = sdk.createShape(geom, material, true);
dynamic->attachShape(box);
PxRigidBodyExt::updateMassAndInertia();
shape->release();

dynamic->setAngularDamping(0.5f);
dynamic->setLinearVelocity(velocity);

gScene->addActor(*dynamic);
```

Actor → Description

API Interface : Actors - Static

2.8.1

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;

NxBoxShapeDesc boxDesc;
boxDesc.dimensions              = NxVec3(1.5f, 1.5f, 1.5f);
actorDesc.shapes.pushBack(&boxDesc);

actorDesc.density               = 1.0f;
actorDesc.body                  = NULL;
actorDesc.globalPose.t          = NxVec3(3.0f, 0.0f, 0.0f);

NxActor *pActor = gScene->createActor(actorDesc);
```

3.3.4

```
PxRigidStatic* static =
                    PxCreateStatic(sdk, trans, geom, *material, 1.0f);

PxRigidStatic* static =
                    sdk>createRigidStatic(trans);

PxShape* box = sdk.createShape(geom, material, true);
dynamic->attachShape(box);
shape->release();

gScene->addActor(*static);
```

API Interface : Actors - Kinematic

2.8.1

```
NxActorDesc actorDesc;

NxBodyDesc bodyDesc;
bodyDesc.flags                  |= NX_BF_KINEMATIC;

NxCapsuleShapeDesc capsuleDesc;
capsuleDesc.radius              = 1.0f;
capsuleDesc.height              = 1.5f;
actorDesc.shapes.pushBack(&capsuleDesc);

actorDesc.density               = 1.0f;
actorDesc.body                  = &bodyDesc;
actorDesc.globalPose.t          = NxVec3(0.0f, 2.0f, 0.0f);

NxActor *pActor = gScene->createActor(actorDesc);
```

3.3.4

```
PxRigidDynamic* kinematic =
                    PxCreateKinematic(sdk, trans, geom, *material, 1.0f);

PxRigidDynamic* kinematic = sdk>createRigidDynamic(trans);
Kinematic->setRigidBodyFlag(PxRigidBodyFlag::eKINEMATIC, true)

PxShape* sphere = sdk.createShape(geom, material, true);
kinematic->attachShape(sphere);

shape->release();

gScene->addActor(*dynamic);
```

Dynamic

Flag

API Interface : Shapes - 생성

2.8.1

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;

NxBoxShapeDesc boxDesc;
boxDesc.dimensions              = NxVec3(1.5f, 1.5f, 1.5f);
actorDesc.shapes.pushBack(&boxDesc);

actorDesc.density               = 1.0f;
actorDesc.body                  = NULL;
actorDesc.globalPose.t          = NxVec3(3.0f, 0.0f, 0.0f);

NxActor *pActor = gScene->createActor(actorDesc);
```

3.3.4

```
PxRigidStatic* actor = sdk>createRigidStatic(trans);

PxShape* box =
    sdk.createShape(PxBoxGeometry(1.0,1.0,1.0), material, true); // true : 공유
actor->attachShape(box);
box->release();

actor->createShape(PxBoxGeometry(1.0, 1.0, 1.0), material);

gScene->addActor(*actor);
```

API Interface : Shapes - 공유 셰이프

2.8.1

3.3.4

```
PxShape* box =
    sdk.createShape(PxBoxGeometry(), material, true); // 공유

box->setLocalPose(pose); // assert
box->setFlag(PxShapeFlag::eSIMULATION_SHAPE, true); // assert
box->setFlag(PxShapeFlag::eSCENE_QUERY_SHAPE, true); // assert

actor1->attachShape(*box); // ok
actor2->attachShape(*box); // ok

PxShape* sphere =
    sdk.createShape(PxShpereGeometry(), material, false); // 비공유

sphere->setLocalPose(pose); // ok
box->setFlag(PxShapeFlag::eSIMULATION_SHAPE, true); // ok
box->setFlag(PxShapeFlag::eSCENE_QUERY_SHAPE, true); // ok

actor1->attachShape(*sphere); // ok
actor2->attachShape(*sphere); // error
```

메모리 절감 효과

API Interface : i3Px - Actor/Shape

2.8.1 + 3.3.4 => i3Px

```
namespace i3Px
{
    struct I3_EXPORT_PX DynamicActorDesc
    {
        RigidBodyDesc rigidBody;

        REAL32 linearDamping;
        REAL32 angularDamping;
        REAL32 maxAngularVelocity;
        REAL32 density;
        UINT32 solverIterationCount;
        REAL32 sleepThreshold;
        REAL32 contactReportThreshold;
        bool isKinematicMode;
    };

    I3_EXPORT_PX void InitializeDynamicActorDesc(DynamicActorDesc& desc);

    class I3_EXPORT_PX DynamicActor : public RigidBody
    {
    public:
        DynamicActor();
        DynamicActor(const DynamicActorDesc& actorDesc, Scene* scene);
        virtual ~DynamicActor();

        virtual void                SetLinearDamping(REAL32 damp);
        virtual REAL32              GetLinearDamping() const;

        virtual void                SetAngularDamping(REAL32 damp);
        virtual REAL32              GetAngularDamping() const;
```

```
            descPx.linearDamping                    = bodyNx.linearDamping;
            descPx.maxAngularVelocity               = bodyNx.maxAngularVelocity;
            descPx.sleepThreshold                   = bodyNx.sleepEnergyThreshold;
            descPx.solverIterationCount             = bodyNx.solverIterationCount;

            // create dynamic actor.
            callback.OnBuildPxDynamicActorDesc(descPx);

            actorPx = &scene->CreateDynamicActor(descPx);
        }
        else
        {
            i3Px::StaticActorDesc descPx;
            i3Px::InitializeStaticActorDesc(descPx);

            descPx.base.name = adescNX.name;
            descPx.base.dominanceGroup = adescNX.dominanceGroup & 0xff; // 3.0 에서는 8비트임에 유의.

            descPx.globalPose = i3Px::ToMatrix(adescNX.globalPose);

            i3::copy(shapePxList.begin(), shapePxList.end(), i3::back_inserter(descPx.shapes));

            callback.OnBuildPxStaticActorDesc(descPx);

            actorPx = &scene->CreateStaticActor(descPx);
        }
```

API Interface : Collision

2.8.1

```
void SetActorCollisionGroup(NxActor *actor, NxCollisionGroup group)
{
    NxU32 nbShapes = actor->getNbShapes();
    NxShape*const* shapes = actor->getShapes();

    while (nbShapes--)
    {
        shapes[nbShapes]->setGroup(group);
    }
}

 SetActorCollisionGroup(box1, 1);
 SetActorCollisionGroup(capsule1, 2);
 SetActorCollisionGroup(sphere1, 2);

 gScene->setGroupCollisionFlag(1, 2, true);
 gScene->setGroupCollisionFlag(2, 3, true);
 gScene->setGroupCollisionFlag(1, 3, false);
```

3.3.4

```
PxShape* box =
     sdk.createShape(PxBoxGeometry(1.0,1.0,1.0), material, true);

box->setSimulationFilterData ... ?????
box->setQueryFilterData ... ?????
```

API Interface : Collision

3.3.4 – PxFilterData ?

```
/**
\brief PxFilterData is user-definable data which gets passed into the collision filtering shader and/or callback.

@see PxShape.setSimulationFilterData() PxShape.getSimulationFilterData() PxSimulationFilterShader PxSimulationFilterCallback
*/
struct PxFilterData
{
//= ATTENTION! =====================================================================================
// Changing the data layout of this class breaks the binary serialization format.  See comments for
// PX_BINARY_SERIAL_VERSION.  If a modification is required, please adjust the getBinaryMetaData
// function.  If the modification is made on a custom branch, please change PX_BINARY_SERIAL_VERSION
// accordingly.
//==================================================================================================

    PX_INLINE PxFilterData(const PxEMPTY&)
    {
    }
    PX_INLINE void setToDefault()
    {
        *this = PxFilterData();
    }

    PxU32 word0;
    PxU32 word1;
    PxU32 word2;
    PxU32 word3;
};
```

```
PxSceneDesc sceneDesc(gPhysics->getTolerancesScale());
sceneDesc.gravity = PxVec3(0.0f, -9.81f, 0.0f);
gDispatcher = PxDefaultCpuDispatcherCreate(2);
sceneDesc.cpuDispatcher = gDispatcher;
sceneDesc.filterShader  = PxDefaultSimulationFilterShader;
gScene = gPhysics->createScene(sceneDesc);

gMaterial = gPhysics->createMaterial(0.5f, 0.5f, 0.6f);
```

API Interface : Collision

3.3.4 – PxFilterData ?

```
PxFilterFlags physx::PxDefaultSimulationFilterShader(
    PxFilterObjectAttributes attributes0,
    PxFilterData filterData0,
    PxFilterObjectAttributes attributes1,
    PxFilterData filterData1,
    PxPairFlags& pairFlags,
    const void* constantBlock,
    PxU32 constantBlockSize)
{
    PX_UNUSED(constantBlock);
    PX_UNUSED(constantBlockSize);

    // let triggers through
    if(PxFilterObjectIsTrigger(attributes0) || PxFilterObjectIsTrigger(attributes1))
    {
        pairFlags = PxPairFlag::eTRIGGER_DEFAULT;
        return PxFilterFlags();
    }

    // Collision Group
    if (!gCollisionTable[filterData0.word0][filterData1.word0]())
    {
        return PxFilterFlag::eSUPPRESS;
    }

    // Filter function
    PxGroupsMask g0 = convert(filterData0);
    PxGroupsMask g1 = convert(filterData1);

    PxGroupsMask g0k0;  gTable[gFilterOps[0]](g0k0, g0, gFilterConstants[0]);
    PxGroupsMask g1k1;  gTable[gFilterOps[1]](g1k1, g1, gFilterConstants[1]);
    PxGroupsMask final; gTable[gFilterOps[2]](final, g0k0, g1k1);

    bool r = final.bits0 || final.bits1 || final.bits2 || final.bits3;
    if (r != gFilterBool)
    {
        return PxFilterFlag::eSUPPRESS;
    }
```

Group : PxFilterData.word0

Filtering : PxFilterData.word2, word3

기대했던 기존 PB 컬리전 실행이 안됨

API Interface : Collision

3.3.4 – PxDefaultSimulationFilterShader

```
PxFilterFlags physx::PxDefaultSimulationFilterShader(
    PxFilterObjectAttributes attributes0,
    PxFilterData filterData0,
    PxFilterObjectAttributes attributes1,
    PxFilterData filterData1,
    PxPairFlags& pairFlags,
    const void* constantBlock,
    PxU32 constantBlockSize)
{
    PX_UNUSED(constantBlock);
    PX_UNUSED(constantBlockSize);

    // let triggers through
    if(PxFilterObjectIsTrigger(attributes0) || PxFilterObjectIsTrigger(attributes1))
    {
        pairFlags = PxPairFlag::eTRIGGER_DEFAULT;
        return PxFilterFlags();
    }

    // Collision Group
    if (!gCollisionTable[filterData0.word0][filterData1.word0]())
    {
        return PxFilterFlag::eSUPPRESS;
    }

    // Filter function
    PxGroupsMask g0 = convert(filterData0);
    PxGroupsMask g1 = convert(filterData1);

    PxGroupsMask g0k0;  gTable[gFilterOps[0]](g0k0, g0, gFilterConstants[0]);
    PxGroupsMask g1k1;  gTable[gFilterOps[1]](g1k1, g1, gFilterConstants[1]);
    PxGroupsMask final; gTable[gFilterOps[2]](final, g0k0, g1k1);

    bool r = final.bits0 || final.bits1 || final.bits2 || final.bits3;
    if (r != gFilterBool)
    {
        return PxFilterFlag::eSUPPRESS;
    }
```

2.8.1 – Filtering.cpp

```
bool Scene::needContacts(const Shape& a, const Shape& b) const  // Special version for compounds, replac
{
    if(!(sceneFlags & SCENE_COLLISIONS))
        return false;

    if(a.getFlagFast(NX_SF_DISABLE_COLLISION) || b.getFlagFast(NX_SF_DISABLE_COLLISION))
        return false;

    if((a.getRbActor().getActorPublicFlags() & NX_AF_DISABLE_COLLISION) || (b.getRbActor().getActorPubli
        return false;

    if(!getU32CollisionFlag(a.getCollisionGroupFast(), b.getCollisionGroupFast()))
        return false;

    const Body* body0 = a.getBodyFast();
    const Body* body1 = b.getBodyFast();

    // Disable collision detection between kinematics (both kin-kin and kin-static):
    //exception: kin-stat and kin-kin triggers are OK
    //exception: compounds containing triggers are OK ---> stuff in compounds will later be filtered age
    NX_ASSERT (!((a.getTypeFast() == NX_SHAPE_COMPOUND) || (b.getTypeFast() == NX_SHAPE_COMPOUND) ));//w

    bool b0isDynamic = body0 && !(body0->getPublicFlagsFast() & NX_BF_KINEMATIC);
    bool b1isDynamic = body1 && !(body1->getPublicFlagsFast() & NX_BF_KINEMATIC);
    if (!(b0isDynamic || b1isDynamic))  //at least one is really dynamic
    {
        if (!(a.isTriggerFast() || b.isTriggerFast()))  //neither is trigger.
            return false;
    }

    // New filtering
    if(!filterFunction(a.getGroupsMaskFast(), b.getGroupsMaskFast()))
        return false;

    // Check pair flags of the shapes.
    if ((getShapePairFlagsFast(a, b) & NX_IGNORE_PAIR) != 0)
        return false;

    // Check pair flags of the actors.
    if ((getActorPairFlags(a.getRbActor(), b.getRbActor()) & NX_IGNORE_PAIR) != 0)
        return false;
```

?

기대했던 기존 PB 컬리전 실행이 안됨

API Interface : Collision

i3Px Wrapping – 2.8.1 Interface 호환

```
namespace Nx    // 2.0 버전 호환 버전.
{
    typedef Flags<INT32, UINT16> FilterGroup16;

    struct I3_EXPORT_PX FilteringGroups
    {
        FilterGroup16 groups0, groups1, groups2, groups3;
    };

    I3_EXPORT_PX void InitializeFilteringGroups(FilteringGroups& src);
    I3_EXPORT_PX bool operator==(const FilteringGroups& filter0, const FilteringGroups& filter1);
    I3_EXPORT_PX bool operator!=(const FilteringGroups& filter0, const FilteringGroups& filter1);

    struct I3_EXPORT_PX Groups
    {
        // collision groups(0 ~ 32). see Lesson 109 - Collision Groups.
        UINT32 group;

        // secondary collision group(filtering masks). see Lesson 110 - Collision Filtering.
        FilteringGroups filtering;

        // reserve.
        UINT32 reserve;
    };
```

```
        I3_EXPORT_PX void   SetGroupCollisionFlag(UINT32 group0, UINT32 group1, bool enable);
        I3_EXPORT_PX bool   GetGroupCollisionFlag(UINT32 gorup0, UINT32 group1);
        I3_EXPORT_PX void   ClearGroupCollisionFlags();

        I3_EXPORT_PX void   SetGroup(Actor& actor, UINT32 group);
        I3_EXPORT_PX UINT32 GetGroup(const Actor& actor);

        I3_EXPORT_PX void   SetGroupsMask(Actor& actor, FilteringGroups groups);

        I3_EXPORT_PX void   SetFilterEquation(const FilterOp::Enum& op0, const FilterOp::Enum& op1, const Filte
                                              const FilteringGroups& K0, const FilteringGroups& K1, bool result

        I3_EXPORT_PX void   SetFilterOps(const FilterOp::Enum& op0, const FilterOp::Enum& op1, const FilterOp::
        I3_EXPORT_PX void   SetFilterBool(bool result);
        I3_EXPORT_PX void   SetFilterConstant(const FilteringGroups& k0, const FilteringGroups& k1);

        I3_EXPORT_PX void   SetActorPairFlags(const Actor& a0, const Actor& a1, PxPairFlags flags);
        I3_EXPORT_PX bool   Get
                            void i3Px::Simulation::Nx::SetActorPairFlags(const i3Px::Actor & a0, const i3Px::Actor & a1, physx::PxPai

        I3_EXPORT_PX void   SetShapePairFlags(const Shape& s0, const Shape& s1, PxPairFlags flags);
        I3_EXPORT_PX bool   GetShapePairFlags(const Shape& s0, const Shape& s1, PxPairFlags& out);

        I3_EXPORT_PX void   SetActorGroupPairFlags(UINT32 groupA0, UINT32 groupA1, PxPairFlags flags);
        I3_EXPORT_PX bool   GetActorGroupPairFlags(UINT32 groupA0, UINT32 groupA1, PxPairFlags& out);

        I3_EXPORT_PX CallbackFunc GetCallback();

        I3_EXPORT_PX PxFilterFlags CheckPairFlags(const PairFound& pair);
    }
}
```

API Interface : Collision

i3Px Wrapping – 2.8.1 Interface 호환

```
PxFilterFlags NxDefaultCallback(PxFilterObjectAttributes attributes0, PxFilterData filterData0,
                                PxFilterObjectAttributes attributes1, PxFilterData filterData1,
                                PxPairFlags& pairFlags,
                                const void* constantBlock, PxU32 constantBlockSize)
{
    // let triggers through
    if (PxFilterObjectIsTrigger(attributes0) || PxFilterObjectIsTrigger(attributes1))
    {
        pairFlags = PxPairFlag::eNOTIFY_TOUCH_FOUND |
                    PxPairFlag::eNOTIFY_TOUCH_LOST |
                    PxPairFlag::eDETECT_DISCRETE_CONTACT |
                    PxPairFlag::eNOTIFY_TOUCH_PERSISTS;

        return PxFilterFlags();
    }

    // Collision Group
    if (!g_NxCollisionTable.GetFlag(filterData0.word0 & 0xffff, filterData1.word0 & 0xffff))
    {
        return PxFilterFlag::eSUPPRESS;
    }

    // filtering.
    FilterData fd0 = ToFilterData(filterData0);
    Groups g0 = ToGroups(fd0);

    FilterData fd1 = ToFilterData(filterData1);
    Groups g1 = ToGroups(fd1);

    if (!g_NxFilteringEquation.operator()(g0.filtering, g1.filtering))
    {
        return PxFilterFlag::eSUPPRESS;
    }

    // callback flags.
    pairFlags = PxPairFlag::eSOLVE_CONTACT |
                PxPairFlag::eDETECT_DISCRETE_CONTACT |
                PxPairFlag::eNOTIFY_TOUCH_FOUND |
                PxPairFlag::eNOTIFY_TOUCH_LOST |
```

+

```
class SceneContactCallback : public i3Px::Callback
{
public:
    virtual ~SceneContactCallback() {}

    virtual PxFilterFlags   OnPairFound(const i3Px::PairFound& pair)
    {
        return i3Px::Simulation::Nx::CheckPairFlags(pair);
    }
};
```

Data Converting