Physics engines

Wolfgang Knecht

Institute of Computer Graphics and Algorithms

Vienna University of Technology

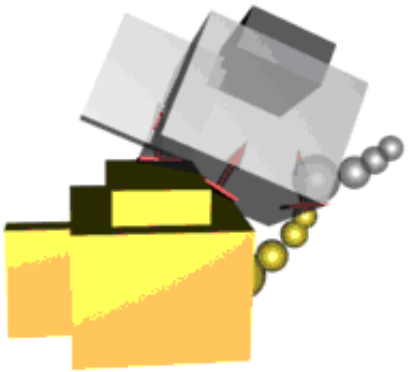
Topics for today

- What is a physics engine?
  - Available engines
- How to use NVIDIA PhysX
  - Create Scene
  - Actors
    - Shapes
    - Dynamic, static and kinematic actors
  - Joints
  - Simulation
  - Debugging

What is a physics engine?

- Handles collision detection
- Physical simulation
- Independent form what you see
  - Low resolution approximation of scene

Graphical representation



Approximation for physical simulation

Available engines

- PhysX
- Havok
- Bullet
  - OpenSource
- ODE
  - OpenSource

PhysX by NVIDIA

Get the PhysX SDK

- Download form NVIDIA Developer website [1]
  - free, but you have to register (takes some time)
- Also get the PhysX SystemSoftware
  - Hardware acceleration on all GeForce 8-series, 9-series and 200-series

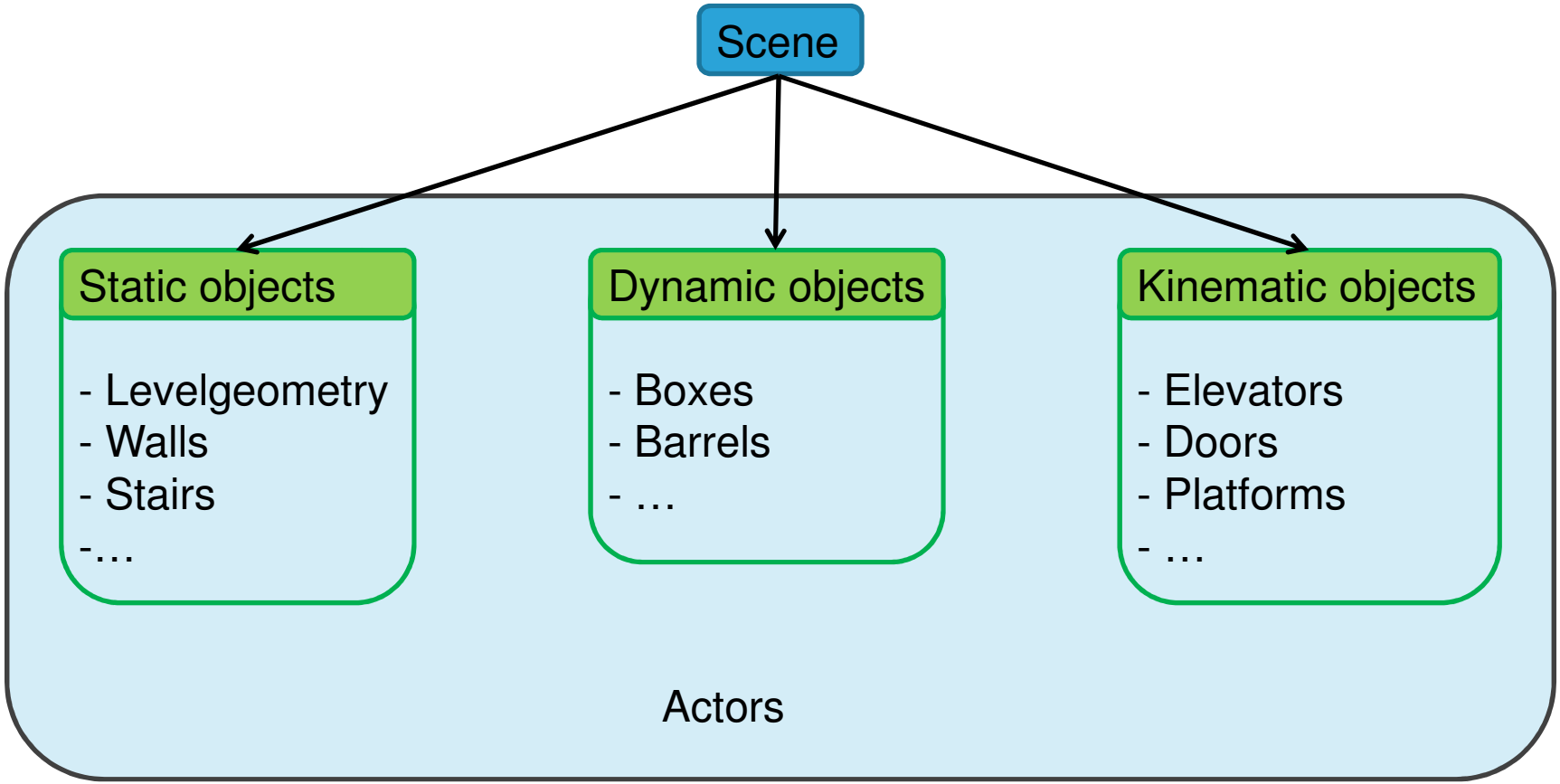
Prepare your project to use the SDK

- Specify include directories
  - '*SDKs\Physics\include*‘
  - '*SDKs\Foundation\include*‘
  - '*SDKs\PhysXLoader\include*‘
  - '*SDKs\Cooking\include*' (optional)
  - '*SDKs\NxCharacter\include*' (optional)
- Identify Library
  - '*SDKs\lib\win32\PhysXLoader.lib*,
- PhysXLoader.dll

Init the SDK

```
#include "NxPhysics.h"

NxPhysicsSDK *gPhysicsSDK =
        NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION, &myAllocator,
        &myOutputStream);

if(!gPhysicsSDK) Error("Wrong SDK DLL version?");


...


gPhysicsSDK->release();
```


The scene and it‘s actors



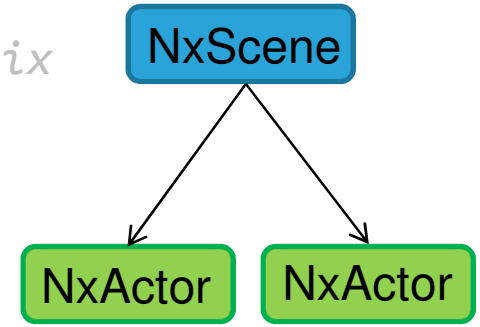
Create a physics scene

NxScene

```
NxSceneDesc sceneDesc;
sceneDesc.gravity.set(0,-9.8f,0);
NxScene *gScene = gPhysicsSDK->createScene(sceneDesc);
if (!gScene)    Error("Can't create scene!");
```


Actors

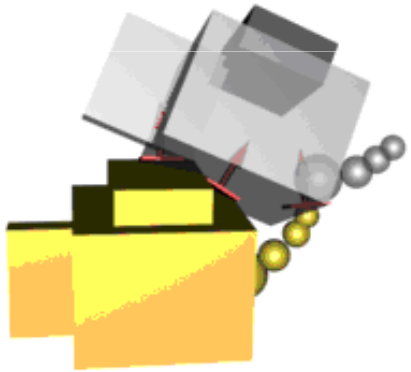
```
NxActorDesc actorDesc;
actorDesc.globalPose = ...; // initial modelmatrix
```

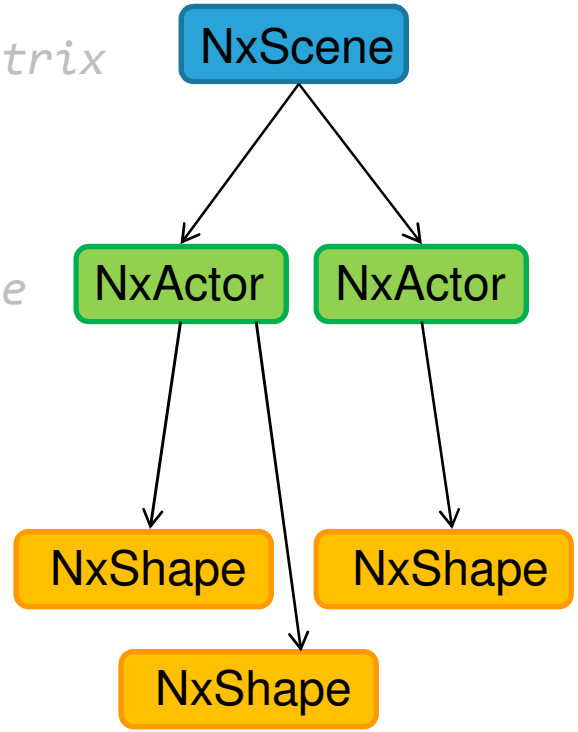



```
gScene->createActor(actorDesc);
```

Actors contain shapes

```
NxActorDesc actorDesc;
actorDesc.globalPose = ...; // initial modelmatrix

NxSphereShapeDesc shapeDesc;
shapeDesc.radius = 2.0;
shapeDesc.localPose = ...; // Pose of the shape
```

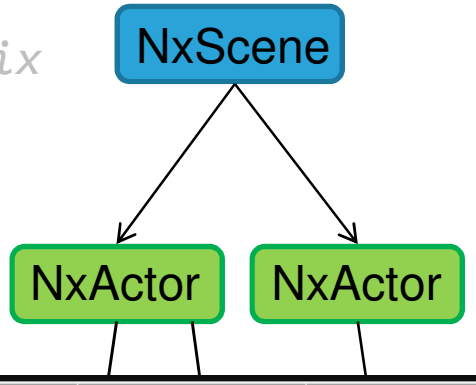





```
actorDesc.shapes.pushBack(&shapeDesc);

gScene->createActor(actorDesc);
```

Actors contain shapes

```
NxActorDesc actorDesc;
actorDesc.globalPose = ...; // initial modelmatrix
```



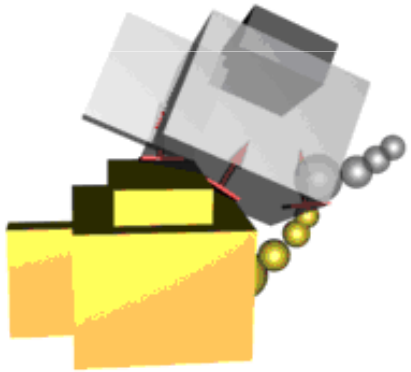

```
NxSphereShapeDesc shapeDesc;
shapeDesc.radius = 2.0;
shapeDesc.localPose = ...; // Pose of the shape
```

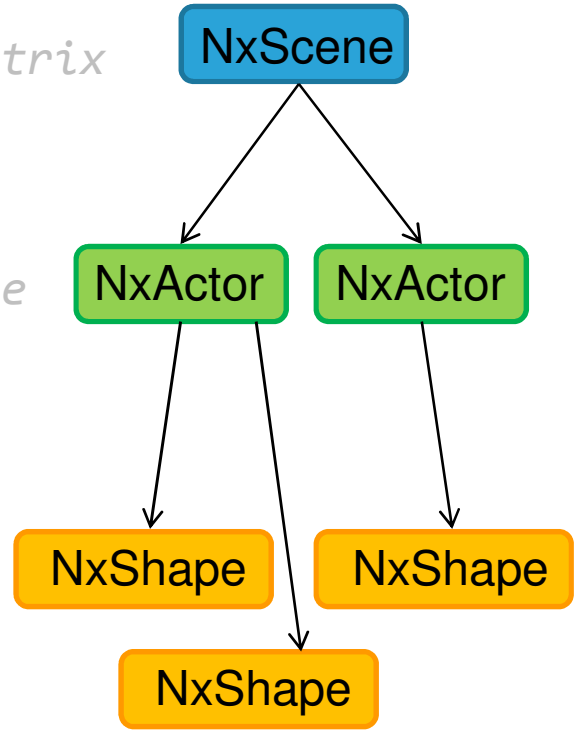
	NxSphereShape	NxBoxShape	NxCapsuleShape	NxPlaneShape	NxConvexShape	NxHeightfieldShape	NxWheelShape	NxTriangleMeshShape	NxTriangleMeshShape (Heightfield)
NxSphereShape	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxBoxShape	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxCapsuleShape	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxPlaneShape	✓	✓	✓	✗	✓	✗	✓	✓	✗
NxConvexShape	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxHeightfieldShape	✓	✓	✓	✗	✓	✗	✓	✗	✗
NxWheelShape	✓	✓	✓	✓	✓	✓	✗	✓	✓
NxTriangleMeshShape	✓	✓	✓	✓	✓	✗	✓	✗	✗
NxTriangleMeshShape (heightfield)	✓	✓	✓	✗	✓	✗	✓	✗	✗

```
actorDesc.shapes.pushBack(&shapeDesc);

gScene->createActor(actorDesc);
```

Actors contain shapes

```
NxActorDesc actorDesc;
actorDesc.globalPose = ...; // initial modelmatrix

NxSphereShapeDesc shapeDesc;
shapeDesc.radius = 2.0;
shapeDesc.localPose = ...; // Pose of the shape
```

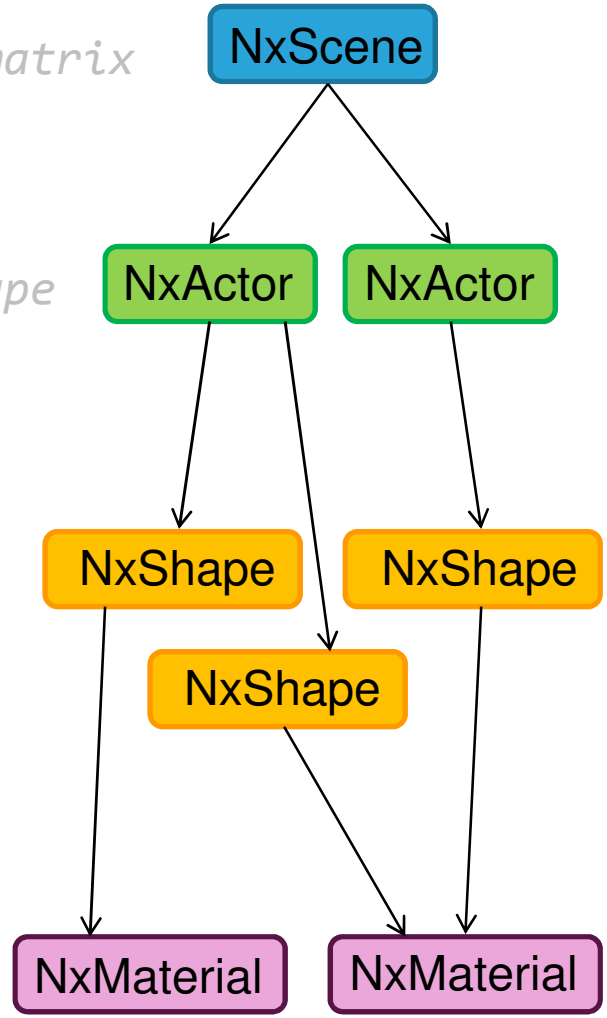





```
actorDesc.shapes.pushBack(&shapeDesc);

gScene->createActor(actorDesc);
```


Shapes have materials

```
NxActorDesc actorDesc;
actorDesc.globalPose = ...; // initial modelmatrix

NxSphereShapeDesc shapeDesc;
shapeDesc.radius = 2.0;
shapeDesc.localPose = ...; // Pose of the shape

NxMaterialDesc materialDesc;
materialDesc.restitution = 0.7f;
materialDesc.staticFriction = 0.5f;
materialDesc.dynamicFriction = 0.5f;

NxMaterial *newMaterial=
        gScene->createMaterial(materialDesc);
shapeDesc.materialIndex=
        newMaterial->getMaterialIndex();

actorDesc.shapes.pushBack(&shapeDesc);

gScene->createActor(actorDesc);
```



Dynamic Actors – Rigid Bodies

- Actor needs a body
- User can add forces and torques

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;

// add some shapes to the actor

bodyDesc.mass=10;

actorDesc.body = &bodyDesc;
actorDesc.globalPose.t = NxVec3(0.0f,10.0f,0.0f);
                                    // set initial position.

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

Static Actors

- Set actor‘s body to NULL

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;

// add some shapes to the actor

bodyDesc.mass=10;

actorDesc.body = &bodyDesc;
actorDesc.globalPose.t = NxVec3(0.0f,10.0f,0.0f);
                                        // set initial position.

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

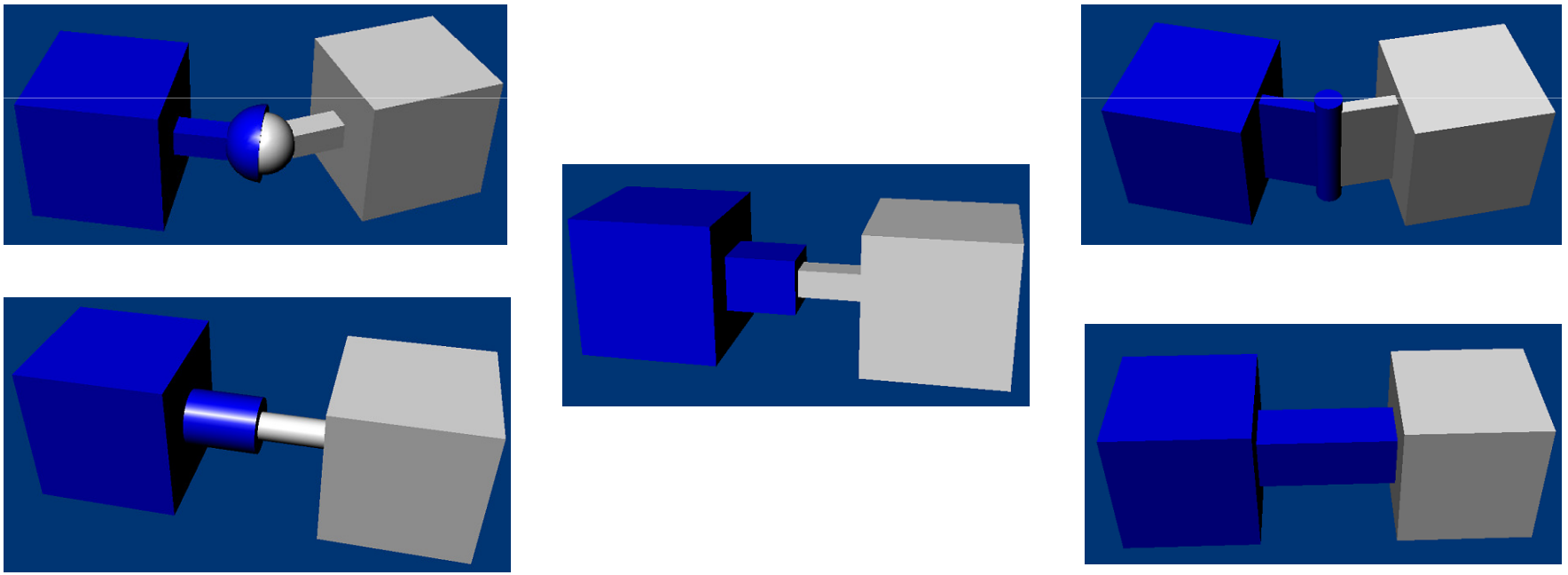
Kinematic Actors

- Does not move in response to forces, gravity, collision impulses, or if tugged by joints
- Moving platforms, elevators, …

```
actor1->raiseBodyFlag(NX_BF_KINEMATIC);

actor1->moveGlobalPose(mat34);
actor1->moveGlobalPosition(vec3);
actor1->moveGlobalOrientation(mat33);

// do NOT use actor1->setGlobal*()
```

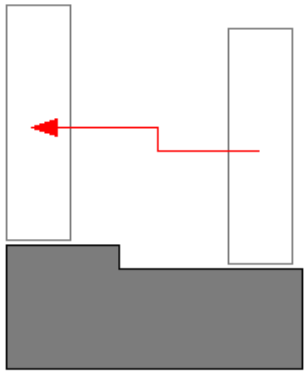
Joints

- Connect two actors
- Several Joint Types
- Motors, Springs and Special Limits



Character Controller

- Controllable kinematic actor
- Only boxes (NxBoxController) and capsules (NxCapsuleController) are supported



```
NxControllerManager* gManager =
        NxCreateControllerManager(myAllocator);
NxCapsuleControllerDesc desc;
desc.radius = 0.5;
desc.height = 2.0;
desc.stepOffset = 0.5;
NxController* c = gManager->createController(scene, desc);

...
c->move(disp, 0xffffffff, 0.000001f, collisionFlags, sharpness);
...

 NxReleaseControllerManager(gManager);
```

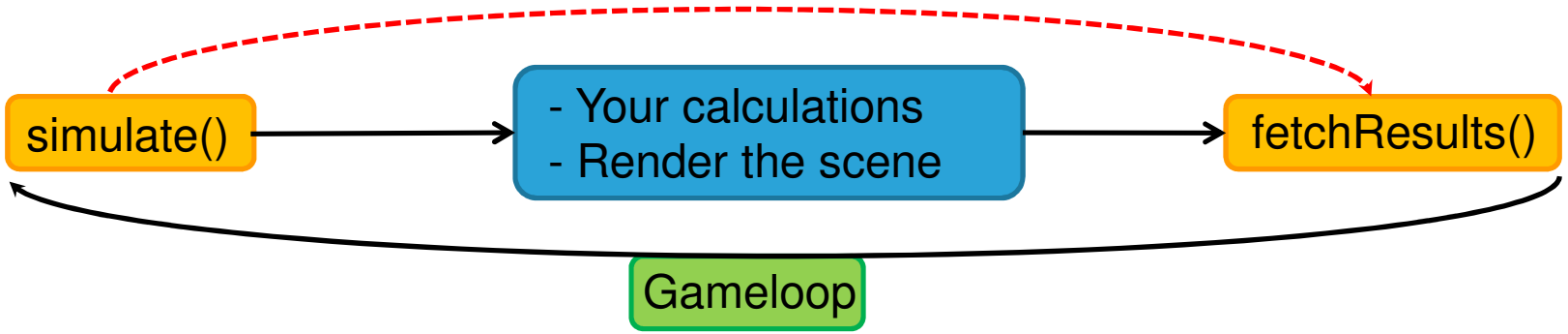
Trigger

- Very useful to trigger events
  - Open a door
  - Start an elevator
  - Change background music

```
shapeDesc.shapeFlags |= NX_TRIGGER_ENABLE;

// myTriggerCallback:
// instance from user defined trigger class
// derived from NxUserTriggerReport

gScene->setUserTriggerReport(&myTriggerCallback);
```

Simulate and fetch results

- Simulations runs in the background




```
gScene->setTiming(1.0f/60.f, 8,NX_TIMESTEP_FIXED);

// gameloop {
        gScene->simulate(elapsedTime);
        gScene->flushStream();

        // do your calculations and the rendering

        gScene->fetchResults(NX_RIGID_BODY_FINISHED, true);

// }
```


Connect PhysX to the Renderer

- Create your **model matrices** using the actor‘s pose

```
actor1->getGlobalPose();              // returns NXMat34

actor1->getGlobalPosition();          // returns NxVec3

actor1->getGlobalOrientation();       // returns NxMat33
```


Cleanup

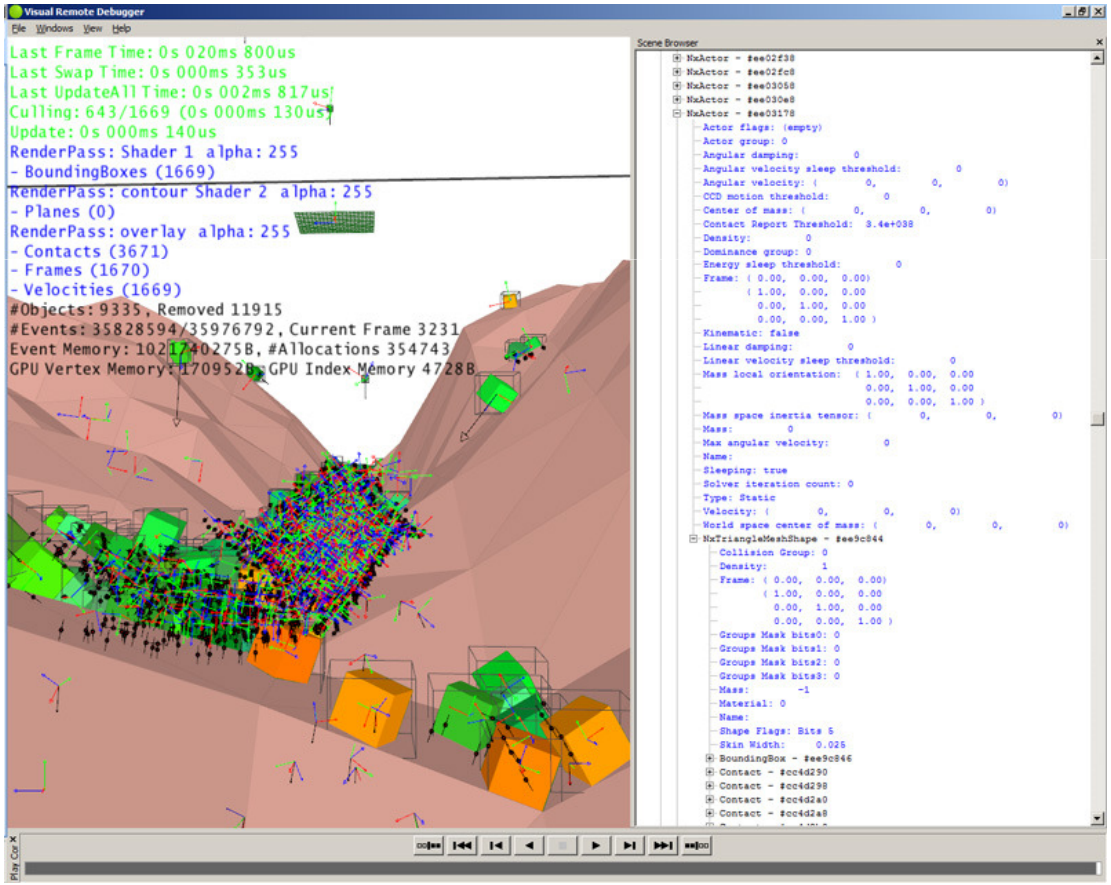
- **Nx…::release…** → releases Child and all associated objects

- **NxScene::releaseActor** → releases Actor and all associated shapes
- **NxPhysicsSDK::releaseScene** → releases Scene and all actors, joints, materials, … created in the scene

- `gPhysicsSDK->release();`

Debugging

- Debug rendering inside your application or
- Visual Debugger [2]



Several more features

- Fluids
- Cloth
- Soft Bodies
- Force Fields
- …

Do you really need a physics engine?

- Physics engines come with new problems
  - Finding right parameters is not easy
  - …
- Creating vehicle physics „by hand“ can be easier than using a physics engine
- Collision detection for simple objects like spheres or boxes can be done with less effort
  - E.g. Labyrinth games need no physics engine

References

- [1] PhysX Developer Zone, http://developer.nvidia.com/object/physx.html
- [2] PhysX Visual Debugger, http://developer.nvidia.com/object/pvd_home.html
- Havok, http://www.havok.com
- Bullet, http://bulletphysics.org
- ODE, http://www.ode.org

The End

- Thanks for your attention!
- Questions?