nV Physics SDK

Programmer's Guide
(first draft)

Table of Contents

Introduction

This document is a first draft development guide for the nV Physics SDK. By the time this draft is being written, the SDK is not yet released, so if you have purchased a version of the SDK, you should not be reading this. However, it can be used as a guide for using any pre-release versions of the SDK.

The nV Physics SDK is mainly intended to be a complete and easy to use physics solution for game development. If you have any comments, suggestions or problems concerning the SDK, please contact us at info@ThePhysicsEngine.com .

Currently the SDK only supports C++ and is distributed in static linkable library format. Different wrappers for language and platform portability are planned in the near future.

Features

The nV physics SDK currently has the following features:

- Collision detection system with broad phase (more than one method) and narrow phase with the following collision primitives:
  - Spheres
  - Boxes
  - Planes
  - Convex Triangle Meshes
  - Concave Triangle Meshes

- Several joint constraints with limits and motors
  - Ball Joints
  - Hinge Joints
  - Double-Hinge Joints
  - Universal Joints
  - Static Joints
  - Stiff Springs
  - Car-Wheel Joints

- Very fast and scalable solving system
- Generic constraint solver with controllable precision
- Auto-deactivation of idle simulation objects.
- Automatic grouping of bodies into subsystems for efficient processing and scalability

Planned features for the future releases:

- Breakable Joints (first release)
- Capsule collision primitive (first release)
- Simple swept collision detection (first release)
- Particle Systems (first release)
- Simple fluid simulation
- More broad-phase collision detection algorithms
- Collision Detection Hierarchies for dynamic triangle meshes
- Water Buoyancy
- Hardware optimization (GPU and PPU utilization)

Starting Physics

First there was nothing, then the programmer created his virtual world, with some clever data structures and a cool renderer, then to set his world into motion, he created an instance of nvPhysicsWorld.

```
#include "nVPhysics.h"

nvPhysicsWorld  MyPhysicsWorld;
```

According to the rules of his world the programmer might set the gravity. And optionally he can specify the maximum number of solver iterations to control precision against performance. It is also recommended to turn on the auto-deactivation of objects.

```
//optional initialization
MyPhysicsWorld.SetGravity(0, -9.8f, 0);
MyPhysicsWorld.SetNoOfSolvingIterations(50);
MyPhysicsWorld.SetObjectsAutoDeactivation(true);
```

Shapes, Primitives, and Bodies

Physical entities (bodies) consist of one or more primitives, each of which is defined by a collision shape and a mass.
This seemingly complex hierarchy allows for greatly flexible combinations. It allows for compound objects, where each object can be composed of multiple primitives, each of which is defined by its mass and a shape structure defining its shape. Moreover, the same shape structure can be used to create multiple primitives and can be used in many objects, this can greatly reduce memory needs especially with complex shapes like convex or concave meshes.

```
nvBody* pBody;

pBody = MyPhysicsWorld.BodyMngr.Create();
//pBody->SetPosition(x, y, z);

nvPrimitive *pPrimitive =  pBody->CreateBox(mass, width,
height, depth);

//optional
pPrimitive->SetElasticity(20);
pPrimitive->SetFriction(30);


pPrimitive =  pBody->CreateSphere(mass, radius);

//optional
pPrimitive->SetElasticity(20);
pPrimitive->SetFriction(30);
```

The same body can have Multiple primitives. Also the same shape can be used for defining multiple primitives.

Collisions, Callbacks, and Broad-Phase

One of the most important aspects to programmers is probably the collision callback, which permits the programmer to define a function to be called whenever two objects collide. It can be used for generating sound effects, explosions, or even preventing the engine from generating collision response between certain object pairs.

```
//define the callback function
static bool CollideCallBack(nvPrimitive* pObj1,  nvPrimitive * pObj2)
{
        If(!collide) return false;
        return true;
}

//set it as the callback function
MyPhysicsWorld.SetCollisionNotify((CollisionNotifyBackFnc)CollideCall
Back);
```

When the callback function returns false, the engine will not process the collision between the two bodies, will not produce a collision response, so the programmer can handle it himself, or else both objects would pass through each other.

The programmer can manually choose from more than one broad-phase collision algorithm.

```
MyPhysicsWorld.SetBroadPhase(SWEEP);
```

Available algorithms are:

- No broad phase! (all against all)
- Simple broad phase (Slightly better and smarter than the first option, eliminates sleeping bodies).
- Sweep and Prune (performs well for most cases)
- Hierarchical grids (hierarchical space hashing)
- Joint Groups (based on contact ad joins data, appropriate when the world bodies are divided in multiple distinct groups).

Forces, Torques, Velocities and beyond

To explicitly affect a certain body with a force (to move) or a torque (to rotate) simply call:

```
pBody->AddOneFrameForce(force.x, force.y, force.z);

//or to specify the point that the force affected:
pBody->AddOneFrameForce(force.x, force.y, force.z, relativePoint.x,
relativePoint.y, relativePoint.z);

//And torque too:
pBody->AddOneFrameTorque( xTorque, yTorque, zTorque);
```

One can also put a permanent force or torque (can be useful for emulating motors), or one can directly set linear or angular velocity. For a complete reference for functions and methods please refer to the nV SDK Reference.

```
pBody->SetAngularVelocity(xAngularVelocity, yAngularVelocity,
zAngularVelocity);
```

Moreover one can directly and explicitly move or rotate an object to a specified location, although it violates physics rules and should be used with caution.

```
pBody->Move(deltaX, deltaY, deltaZ);

pBody->RotateAxisAngle(xAxis, yAxis, zAxis, angle);
```

Joints, Motors, and Limits

To bind two objects with a common joint you simply call:

```
MyPhysicsWorld.JointMngr.CreateStaticJoint(pBody1, pBody2);
```

Supplying it two pointers two the concerned bodies.

Or for example:

```
nvLimitedBallJoint* pRagDollNeckJoint =
MyPhysicsWorld.JointMngr.CreateLimitedBallJoint(pRagDollTorsoBody,
pRagDollHeadBody,headPosition.x,headPosition.y,headPosition.z,
axis1.x, axis1.y, axis1.z, axis2.x, axis2.y, axis2.z);
```

Motors and limits can be set and controlled as follows:

```
//setting limits:
pRagDollNeckJoint->SetMaxLimit(0,PI/4.0f);
pRagDollNeckJoint->SetMinLimit(0,-PI/4.0f);
```

```
//Setting the maximum torque (or force) with which
//a joint can affect bodies.
pRagDollNeckJoint->SetMaxTorque(0,1000.0f);


//Then to operate the motor you specify the speed
//(angular velocity) the joint should move with.
//The joint will try to reach this velocity
//constrained with its maximum torque.
pRagDollNeckJoint->SetRequiredVelocity(0,PI/1.0f);

//The first "0" indicates the axis about which this option is set.
```

To know more about limited-ball joints look it up in the SDK reference.

Running the Simulation

To actually run the simulation one must eventually call:

```
MyPhysicsWorld.RunStep(dt);
```

Supplying the time step value in seconds.

Getting Body States

To ultimately get the current position and orientation of a body the functions:

```
pBody->GetPositon(x,y,z);
float * rot = pBody->GetRotationMatrix();
//returns a 3x3 rotation matrix.
```

Other Resources

nV SDK Function Reference
nV SDK Tutorials
nV SDK Demos Source Code

Known Issues and Future Enhancements

- Convex and concave collision detection algorithms are still undergoing heavy optimization phases and have currently a relatively poor performance in relation to the general performance of the engine.

- There is a problem in resting when objects greatly vary in their densities.

Contact Us

Please if you have any comments, suggestions or problems concerning the SDK, contact us at info@ThePhysicsEngine.com .

Wednesday, July 05, 2006