Práctica 5: Sólidos Rígidos

En esta práctica vamos a usar *PhysX* para realizar la simulación y ver cómo se comporta el sistema profesional de física de *Nvidia*. Con su uso veréis que tiene una estructura muy parecida a la que hemos implementado en nuestros sistemas anteriores. Para consultar la documentación completa de NVIDIA está disponible este enlace:

https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/RigidBodyDynamics.html

Para empezar, tenemos dos tipos de sólidos rígidos, los dinámicos y los estáticos. Los estáticos están pensados para no tener velocidad/aceleración (equivalente a un sólido de masa infinita). Los dinámicos son aquellos que sí que podrán tener velocidad y aceleración. Las clases para gestionarlos son: *PxRigidStatic* y *PxRigidDynamic*.

Para crear uno estático usaremos: *gPhysics->createRigidStatic()* con un *PxTransform* como parámetro.

Para crear uno dinámico llamaremos a: *gPhysics->createRigidDynamic()* también con un objeto *PxTransform* como parámetro.

En cualquier de los dos casos llamaremos a la función miembro *attachShape()* para unir el sólido rígido con un objeto *PxShape*, que podemos crear como hacíamos hasta ahora en nuestras prácticas. Tened en cuenta que, a parte de la forma, el objeto *PxShape* tiene información del material del que está hecho el sólido rígido, dicho material nos ajusta las propiedades de fricción entre superficies y el coeficiente de restitución cuando se produce un choque. Al inicio del archivo main.cpp se genera un material a modo de ejemplo, lo cual se hace con el siguiente método de la clase *PxPhysics*: *PxPhysics::createMaterial(staticFriction, dynamicFriction, restitution);*

Tendremos que ir definiendo sus parámetros dinámicos (masa y tensor de inercia). Para ello haremos uso del método: *PxRigidBodyExt::updateMassAndInertia*. Con esta función especificamos su densidad, que junto con el volumen definido mediante las dimensiones del PxShape, permite calcular la masa y el tensor de inercia de forma automática. También se pueden generar de manera manual usando el método *new_solid->setMassSpaceInertiaTensor({Ixx,Iyy,Izz}).* En el Anexo de este documento se adjunta una tabla para el cálculo rápido de momentos de inercia para formas sencillas.

Una vez creados los sólidos rígidos hay que añadirlos a la escena mediante este método: *gScene->addActor()*. Pasando como parámetro el sólido rígido creado. Esto es necesario para que puedan interactuar entre sí.

Para que se puedan visualizar será necesario crear un *RenderItem*, igual que en prácticas anteriores, solo que en lugar de un *Transform* le pasaremos la instancia de sólido rígido que hemos creado.

En *PhysX* la gravedad se trata como algo separado. Se puede configurar de forma global para toda la escena poniendo un valor de Vector3 en esta variable: *sceneDesc.gravity*. Esto debe hacerse en *initPhysics* antes de la llamada a *gScene = gPhysics->createScene(sceneDesc);* De lo contrario no se tendrá en cuenta.

Actividad 1: Sistema de sólidos con gravedad

Para empezar, adaptaremos los generadores de partículas que hicimos para que en lugar de generar partículas genere sólidos rígidos (gestionados por *PhysX)*. Pondremos un máximo de elementos generados, más restrictivo que con las partículas, para no saturar el sistema y haremos que estos sólidos se vean afectados por la gravedad. También crearemos sólidos estáticos en la escena.

Tareas

- Añadir un sólido rígido estático que sea el suelo.
- Generar una nueva clase "Sistema Solidos" que tenga una funcionalidad similar a la de Sistema de Partículas, pero en lugar de partículas trabaje con Sólidos rígidos.
- Adaptar o generar una nueva clase de generadores de partículas que sean capaces de generar sólidos rígidos. Estos sólidos deben verse afectados por la gravedad y deben de tener diferentes momentos de inercia.
- Genera diferentes tipos de sólidos rígidos con diferentes tipos de materiales cambiando su propiedad de fricción y elasticidad.
- [Opcional] Definir a mano los momentos de inercia de al menos un sólido rígido (de manera aproximada) utilizando la tabla en los anexos.

Actividad 2. Fuerzas y torques aplicados a sólidos.

El siguiente paso es adaptar generador de fuerzas para que se aplique también sobre los sólidos rígidos. Para aplicar una fuerza sobre un sólido rígido la clase *PxRigidDynamic* dispone del método *addForce()*, a la que se pasa un Vector3 con la fuerza (dirección e intensidad). También podéis usar el método *addTorque()* para aplicar un par de fuerzas a un objeto, modificando el momento angular del sólido. Esto se deja como tarea optativa.

Tarea

- Modificar el código, si es necesario, para que las fuerzas generadas por los generadores de fuerzas ya definidos puedan ser aplicadas a los sólidos rígidos. Usar las funciones disponibles en este enlace.
- [Opcional] Probar a generar un nuevo generador de fuerzas que genere pares de fuerza (torques).

Recursos adicionales

Además de lo planteado en las tareas anteriores los sólidos rígidos de physX presentan funcionalidades adicionales que os pueden resultar muy útiles a la hora de diseñar vuestro juego. Se recomienda echarle un vistazo a la documentación del enlace al principio de este documento. Algunas de las funcionalidades más usadas son:

Axis locking (bloqueo de una dimensión)

Esta funcionalidad impide que el sólido rígido se mueva a través de una dimensión determinada. Especialmente útil para juegos en 2D. Tenéis un ejemplo de uso aquí

Rotación de Vectores mediante Cuaternions

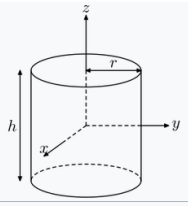
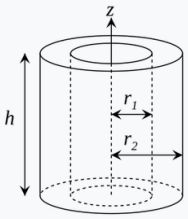
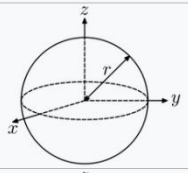
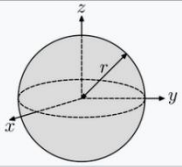
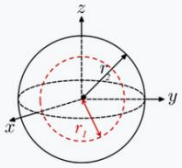
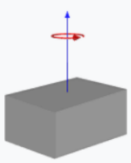
Usar cuaterniones nos permite definir de manera exacta y sencilla rotaciones sobre vectores (cambios de dirección). Internamente Physx usa cuaterniones para definir rotaciones, saber trabajar con cuaterniones nos permite entender cómo se gestionan internamente estas

rotaciones y entender el código base. En este enlace tenéis documentación de como se implementan los cuaterniones en Physx. Un ejemplo de como usar un Cuaternion para rotar un vector “Dir” es el siguiente:

PxQuat qx(nx,ny,nz,nw);

Dir = qx.rotate(Dir);

Anexo: Lista de momentos de inercia directos

Solid cylinder of radius r, height h and mass m. This is a special case of the thick-walled cylindrical tube, with $r_1 = 0$.		$I_z = \frac{1}{2}mr^2$ [1] $I_x = I_y = \frac{1}{12}m\left(3r^2 + h^2\right)$
Thick-walled cylindrical tube with open ends, of inner radius r_1, outer radius r_2, length h and mass m.		$I_z = \frac{1}{2}m\left(r_2^2 + r_1^2\right) = mr_2^2\left(1 - t + \frac{t^2}{2}\right)$ [1] [2] where $t = (r_2 - r_1)/r_2$ is a normalized thickness ratio; $I_x = I_y = \frac{1}{12}m\left(3\left(r_2^2 + r_1^2\right) + h^2\right)$ [citation needed] The above formula is for the xy plane passing through the center of mass, which coincides with the geometric center of the cylinder. If the xy plane is at the base of the cylinder, i.e. offset by $d = \frac{h}{2}$, then by the parallel axis theorem the following formula applies: $I_x = I_y = \frac{1}{12}m\left(3\left(r_2^2 + r_1^2\right) + 4h^2\right)$
With a density of ρ and the same geometry		$I_z = \frac{\pi\rho h}{2}\left(r_2^4 - r_1^4\right)$ $I_x = I_y = \frac{\pi\rho h}{12}\left(3(r_2^4 - r_1^4) + h^2(r_2^2 - r_1^2)\right)$
Hollow sphere of radius r and mass m.		$I = \frac{2}{3}mr^2$ [1]
Solid sphere (ball) of radius r and mass m.		$I = \frac{2}{5}mr^2$ [1]
Sphere (shell) of radius r_2 and mass m, with centered spherical cavity of radius r_1. When the cavity radius $r_1 = 0$, the object is a solid ball (above). When $r_1 = r_2$, $\frac{r_2^5 - r_1^5}{r_2^3 - r_1^3} = \frac{5}{3}r_2^2$, and the object is a hollow sphere.		$I = \frac{2}{5}m \cdot \frac{r_2^5 - r_1^5}{r_2^3 - r_1^3}$ [1]
Solid rectangular cuboid of height h, width w, and depth d, and mass m. [7] For a similarly oriented cube with sides of length s, $I_{\mathrm{CM}} = \frac{1}{6}ms^2$		$I_h = \frac{1}{12}m\left(w^2 + d^2\right)$ $I_w = \frac{1}{12}m\left(d^2 + h^2\right)$ $I_d = \frac{1}{12}m\left(w^2 + h^2\right)$

Fuente: https://en.wikipedia.org/wiki/List_of_moments_of_inertia