Tutorial de Física

Sumario

Ver los anteriores tutoriales básicos para más información sobre la creación básica de objetos, manejo del reloj, jerarquía de fotogramas, animaciones, cámaras & vistas, música & sonido y efectos(FXs).

Este tutorial muestra como añadirle propiedades físicas a los objetos y manejar colisiones.

Como puedes ver, las propiedades físicas son completamente manipuladas por datos. Así, creando un objeto con propiedades físicas (ej. con un cuerpo) o sin ellas, resulta en exactamente la misma linea de código.

Los objetos pueden ser enlazados a un cuerpo, que puede ser estático o dinámico.
Cada cuerpo puede estar hecho de hasta 8 partes.

Un cuerpo es definido por:

- su forma (actualmente caja, esfera y malla(ej. poligono convexo) son los únicos disponibles)
- información acerca del tamaño de la forma (esquinas para la caja, centro y radio para la esfera, vértices para la malla)
- si el tamaño de los datos no es especificado, la forma tratará de llenar el cuerpo completo (usando el tamaño y la escala del objeto)
- las banderas propias de colisión definen esta parte
- la máscara de chequeo de colisión define con que otra parte ella puede colisionar [1)]
- una bandera sólida(`Solid`) especificando si esta forma puede solo brindar información acerca de colisiones o si puede impactar en simulaciones de cuerpos físicos (rebota, etc...)
- varios atributos como son restitución, fricción, densidad, ...

En este tutorial creamos muros estáticos sólidos alrededor de la pantalla. Entonces reproducimos cajas en el medio.
El número de cajas creadas serán ajustadas a través del fichero de configuración y es 100 por defecto.

La única interacción posible es usando los botones izquierdo y derechos del ratón (o las teclas izquierda y derecha) para rotar la cámara.
Como mismo lo rotamos, podemos actualizar el vector de gravedad de nuestra simulación.
Haciendo eso, nos da la impresión de que las cajas siempre están cayendo contra el fondo de nuestra pantalla, no importa como la cámara es rotada.

Registramos también los eventos físicos para añadir un FX visual en los dos objetos que colisionan.
Por defecto el FX es un parpadeo de color rápido y es, como norma, ajustable en tiempo real (ej. recargando la configuración histórica aplicaremos los nuevos ajustes inmediatamente si el FX no se mantiene en la cache por defecto).

Actualizando la escala de un objeto (incluso cambiando su escala con FXs) actualizaremos sus propiedades físicas (ej. su cuerpo).
Ten en mente que escalando un objeto con cuerpo físico es más caro que si tenemos que eliminar la

correspondiente forma y recreándolo en tamaño correcto.
Esto está hecho de esta manera ya que nuestro correspondiente plugin físico está basado en Box2D, que no permite rescalado en tiempo real de formas.

Este tutorial solo nos muestra un control básico de físicas y colisiones, pero, por ejemplo, tú puede también ser notificado con eventos por objetos separados o mantener el contacto.

Detalles

Como es usual, empezamos por cargar nuestro fichero de configuración, obteniendo el reloj principal y registrando nuestra función `Update` a el y, por último, por crear nuestro soldado y objectos de la caja.
Por favor, referirse a los tutoriales anteriores para más detalles.

Creamos también nuestros muros. Actualmente no los crearemos uno por uno, los agrupamos en una lista de hijos(`ChildList`) como objeto padre.

```
orxObject_CreateFromConfig("Walls");
```

Esto luce como si hubiéramos creado un solo objeto llamado `Muros` (`Walls`), pero como vemos en el fichero de configuración, el es actualmente un contenedor que reproducirá varios muros.

Por último, creamos nuestras cajas.

```
for(i = 0; i < orxConfig_GetU32("BoxNumber"); i++)
{
  orxObject_CreateFromConfig("Box");
}
```

Como puedes ver, no especificamos nada respecto a las propiedades físicas de nuestros muros o cajas, eso es enteramente hecho en el fichero de configuración y completamente manejado por datos.

Registramos entonces los eventos físicos.

```
orxEvent_AddHandler(orxEVENT_TYPE_PHYSICS, EventHandler);
```

Nada realmente nuevo hasta aquí, miremos directamente a nuestra llamada de retorno `EventHandler`.

```
if(_pstEvent->eID == orxPHYSICS_EVENT_CONTACT_ADD)
{
  orxOBJECT *pstObject1, *pstObject2;

  pstObject1 = orxOBJECT(_pstEvent->hRecipient);
  pstObject2 = orxOBJECT(_pstEvent->hSender);

  orxObject_AddFX(pstObject1, "Bump");
  orxObject_AddFX(pstObject2, "Bump");
```

```
}
```

Básicamente, solo manejamos el nuevo evento contactado y añadimos un FX llamado Bump en ambos objetos colisionadores. Este FX los hace parpadear en un color aleatorio.

Veamos ahora nuestra función `Update`.

```
void orxFASTCALL Update(const orxCLOCK_INFO *_pstClockInfo, void
*_pstContext)
{
  orxFLOAT fDeltaRotation = orxFLOAT_0;

  if(orxInput_IsActive("RotateLeft"))
  {
    fDeltaRotation = orx2F(4.0f) * _pstClockInfo->fDT;
  }
  if(orxInput_IsActive("RotateRight"))
  {
    fDeltaRotation = orx2F(-4.0f) * _pstClockInfo->fDT;
  }

  if(fDeltaRotation != orxFLOAT_0)
  {
    orxVECTOR vGravity;

    orxCamera_SetRotation(pstCamera, orxCamera_GetRotation(pstCamera) +
fDeltaRotation);

    if(orxPhysics_GetGravity(&vGravity))
    {
      orxVector_2DRotate(&vGravity, &vGravity, fDeltaRotation);
      orxPhysics_SetGravity(&vGravity);
    }
  }
}
```

Como puedes ver, obtenemos la actualización desde las entradas `RotarIzquierda`(`RotateLeft`) y `RotarDerecha`(`RotateRight`).
Si una rotación necesita ser aplicada, entonces actualizamos nuestra cámara con `orxCamera_SetRotation()` y actualizamos nuestra simulación de gravedad física en consecuencia.
De esta manera, nuestras cajas siempre parecerán que caen al fonde de nuestra pantalla, sin importar la rotación de la cámara.
Note el uso de `orxVector_2DRotate()` para rotar el vector de gravedad.

PD: Todas las rotaciones en el código de orx siempre son expresadas en radianes!

Veamos ahora a nuestros datos de configuración. Tú puedes encontrar más información de los parametros de configuración en sección cuerpo de los ajustes de configuración.
Primero, creamos implicitamente muchos muros usando la propiedad `ListaDeHijos`(`ChildList`). Vemos debajo como está hecho.

```
[Walls]
ChildList = Wall1 # Wall2 # Wall3 # Wall4; # Wall5 # Wall6
```

Como podemos ver, nuestro objeto Muros(Walls) está vacio, el creará Wall1, Wall2, Wall3 y Wall4 (nota el ';' finalizando la lista aquí).
Puedes remover este ';' para crear 2 muros adicionales.

Veamos ahora como definimos nuestro muro y sus propiedades físicas.
Comencemos con la forma que usaremos para ambos WallBody y BoxBody.

```
[FullBoxPart]
Type        = box
Restitution = 0.0
Friction    = 1.0
SelfFlags   = 0x0001
CheckMask   = 0xFFFF
Solid       = true
Density     = 1.0
```

Aquí le pedimos una parte que usaremos en una forma de caja con no Restitución(Restitution) (ej. no rebote) y alguna Fricción(Friction).
También definimos el SelfFlags y CheckMask para este muro.
La primera define la marca de identidad para esta parte, y la segunda define para que marca es sensitiva (ej. con quien el colisiona).
Básicamente, si tenemos dos objetos: Object1 y Object2. Ellos colisionan si la siguiente expresión es TRUE.

```
(Object1.SeflFlags & Object2.CheckMask) && (Object1.CheckMask &
Object2.SelfFlags)
```

PD: Como no especificamos los atributos TopeIzquierdo(TopLeft) y DebajoDerecho(BottomRight) para esta parte FullBoxPart, el usará el tamaño completo de nuestro cuerpo/objeto a que hace referencia.

Ahora necesitamos definir nuestros cuerpos para las cajas y los muros.

```
[WallBody]
PartList = FullBoxPart

[BoxBody]
PartList  = FullBoxPart
Dynamic   = true
```

Podemos ver que ambos usan la misma parte[2)].
Como Dinámico(Dynamic) está marcada como verdadero(true) para el CuerpoCaja(BoxBody), este objeto se moverá acorde a la simulación física.
Para el CuerpoMuro(WallBody), nada es definido para el atributo Dynamic, es marcado por defecto como falso(false), y los muros no se moverán sin importar que les suceda.

PD: Como no puede haber ninguna colisión entre dos objetos no-dinámicos (ej. estático), los muros no

colisionan incluso si se tocan o superponen. Ahora que tenemos nuestros cuerpos, veamos como los aplicamos a nuestros objetos.
Primero, nuestras cajas. <code ini>[Box] Graphic = BoxGraphic Position = (50.0, 50.0, 0.0) ~ (750.0, 550.0, 0.0) Body = BoxBody Scale = 2.0</code> Como puedes ver, nuestra `Caja(Box)` tiene un atributo `Cuerpo(Body)` puesto en `BoxBody`.
Podemos notar también su posición aleatoria, que significa que siempre que creamos una nueva caja, el tendrá una nueva posición aleatoria en ese rango. Veamos ahora nuestros muros. <code ini>[WallTemplate] Body = WallBody [VerticalWall@WallTemplate] Graphic = VerticalWallGraphic; Scale = @VerticalWallGraphic.Repeat; [HorizontalWall@WallTemplate] Graphic = HorizontalWallGraphic; Scale = @HorizontalWallGraphic.Repeat; [Wall1@VerticalWall] Position = (0, 24, 0) [Wall2@VerticalWall] Position = (768, 24, 0) [Wall3@HorizontalWall] Position = (0, -8, 0) [Wall4@HorizontalWall] Position = (0, 568, 0) [Wall5@VerticalWall] Position = (384, 24, 0) [Wall6@HorizontalWall] Position = (0, 284, 0)</code> Veamos ahora como usamos la herencia una vez más. Primero definimos una `PlantillaMuro(WallTemplate)` que contiene nuestro `WallBody` con un atributo `Body`.
Entonces cuando heredamos desde esa sección con `MuroHorizontal(HorizontalWall)` y `MuroVertical(VerticalWall)`. Ellos tienen la misma propiedad física pero un atributo `Gráfico(Graphic)` diferente.
Ahora que tenemos nuestras plantillas de muros para ambos vertical y horizontal muros, solo necesitamos especificarlos un poco más añadiendo una posición.
Eso que hacemos con `Wall1`, `Wall2`, etc... ===== Recursos =====

1)
dos partes en el mismo cuerpo nuncá colisionarán
2)
ellos pueden tener hasta 8 partes, pero solo 1 es usada aquí

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/es/orx/tutorials/physics?rev=1330803036

Last update: **2025/09/30 17:26 (14 months ago)**