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.

We also creates our walls. Actually we won't create them one by one, we'll group them in a `ChildList` of a parent object.

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

This looks like we only create one object called `Walls`, but as we'll see in the config file, it's actually a container that will spawn a couple of walls.

Lastly, we create our boxes.

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

As you can see, we don't specify anything regarding the physics properties of our walls or boxes, this is entirely done in the config file and is fully data-driven.

We then register to physics events.

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

Nothing really new here, so let's have a look directly to our `EventHandler` callback.

```
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");
}
```

Basically we only handle the new contact event and we add a FX called Bump on both colliding objects. This FX will make them flash in a random color.

Let's now see our Update function.

```
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);
    }
  }
}
```

As you can see, we get the rotation update from the RotateLeft and RotateRight inputs.
If a rotation needs to be applied, we then update our camera with orxCamera_SetRotation() and we update our physics simulation gravity accordingly.
This way, our boxes will always look like they fall toward the bottom of our screen, whichever the camera rotation is.
Note the use of orxVector_2DRotate() so as to rotate the gravity vector.

NB: All rotations in orx's code are always expressed in radians!

Recursos

[1)]
dos partes en el mismo cuerpo nuncá colisionarán