Tutorial de Efectos

Sumario

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

Este tutorial muestra que son los efectos(FXs) y como se crean.

FXs están basados en una combinación de curvas[1)] aplicadas con diferentes parámetros como son escala, rotación, posición, velocidad, transparencia(alpha) y color.

Los FXs se establecen a travez del fichero de configuración requiriendo solamente una linea de código para aplicarselos a un objeto.
Pueden existir hasta 8 curvas de cualquier tipo combinadas para formar un solo FX.
Hasta 4 FXs pueden ser aplicados en el mismo objeto al mismo tiempo.

Los FXs pueden usar valores relativos o absolutos, dependiendo de el atributo `Absolute` en su configuración.
El control sobre la curva de periodo, fase y amplificación con el tiempo también está también garantizada.
La posición y velocidad de los FXs, sus valores de salida pueden usar la orientación y/o escala con que fueron aplicados relativamente en sus respectivos objetos.
Esto permite la creación de unos muy elaborados y bonitos FXs.

Los parámetros del FX pueden ser ajustados en el fichero de configuración y recargados en el aire(tiempo de ejecución) usando la tecla `BackSpace`, a menos que el FX sea especificado en el cache de la memoria(ej. en el atributo `KeepInCache`).
Por ejemplo no puedes ajustar en el aire un circulo FX cuando ya fue definido con ese atributo en el fichero de configuración por defecto.
Todos los demás FXs pueden ser mejorados mientras corre el tutorial.

Como siempre, los parámetros aleatorios pueden ser usados para permitir alguna variedad para los FXs.
Por ejemplo, la escala de oscilaciones, el color del flash y el FXs del movimiento de `attack` son usando valores aleatorios limitados.

Registramos también los eventos del FX a mostrar cuando los FXs son reproducidos y detenidos.
Como el FX es reproducido en la caja de objeto es identificado como cíclico, el nunca para. Por lo tanto el correspondiente evento (`orxFX_EVENT_STOP`) nunca se enviará.

También mostramos brevemente como añadir algún dato personal del usuario a un `orxOBJECT` (aquí una estructura conteniendo un boolean sencillo).
Lo recuperamos en un evento retorno de llamada (event callback) para bloquear el objeto cuando un FX comienza y lo desbloqueamos cuando el FX se detiene.
Usamos este bloqueo para permitir solo un FX a la vez en el soldado.
Solo descrito aquí para fines didácticos.

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.

Registramos entonces los eventos de entrada y efectos(FXs).

```
orxEvent_AddHandler(orxEVENT_TYPE_FX, EventHandler);
orxEvent_AddHandler(orxEVENT_TYPE_INPUT, EventHandler);
```

Como puedes ver, estamos usando la misma llamada de retorno(`EventHandler`) para ambos tipos de eventos.

Echemos un vistazo rápido a la estructura de datos de nuestro `objecto`.

```
typedef struct MyObject
{
  orxBOOL bLock;
} MyObject;
```

Y veamos ahora como se unen a nuestro soldado usando `orxObject_SetUserData()`.

```
MyObject *pstMyObject;

pstMyObject = orxMemory_Allocate(sizeof(MyObject), orxMEMORY_TYPE_MAIN);
pstMyObject->bLock = orxFALSE;

orxObject_SetUserData(pstSoldier, pstMyObject);
```

Necesitamos ver ahora como aplicamos FXs(efectos) en nuestra función `Update`.

```
orxSTRING zSelectedFX;

if(orxInput_IsActive("SelectWobble"))
{
  zSelectedFX = "WobbleFX";
}
else if(orxInput_IsActive("SelectCircle"))
{
  zSelectedFX = "CircleFX";
}

[...]

// No está bloqueado el soldado?
if(!((MyObject *)orxObject_GetUserData(pstSoldier))->bLock)
{
  if(orxInput_IsActive("ApplyFX") && orxInput_HasNewStatus("ApplyFX"))
```

```
  {
    orxObject_AddFX(pstSoldier, zSelectedFX);
  }
}
```

Podemos ver como obtenemos nuestros datos asociados usando `orxObject_GetUserData()` y como se le añade un FX a nuestro soldado, haciendolo de la misma manera de añadiendo un sonido con `orxObject_AddFX()`.

Echemos un vistazo a nuestra función `EventHandler`.
Primero la parte de manejar la entrada, donde solo mostraremos que teclas están siendo usadas por cada entrada activa.

```
if(_pstEvent->eType == orxEVENT_TYPE_INPUT)
{
  if(_pstEvent->eID == orxINPUT_EVENT_ON)
  {
    orxINPUT_EVENT_PAYLOAD *pstPayload;

    pstPayload = (orxINPUT_EVENT_PAYLOAD *)_pstEvent->pstPayload;

    if(pstPayload->aeType[1] != orxINPUT_TYPE_NONE)
    {
      orxLOG("[%s] triggered by '%s' + '%s'.", pstPayload->zInputName,
orxInput_GetBindingName(pstPayload->aeType[0], pstPayload->aeID[0]),
orxInput_GetBindingName(pstPayload->aeType[1], pstPayload->aeID[1]));
    }
    else
    {
      orxLOG("[%s] triggered by '%s'.", pstPayload->zInputName,
orxInput_GetBindingName(pstPayload->aeType[0], pstPayload->aeID[0]));
    }
  }
}
```

Como puedes ver, mostramos una información dependiendo si se usa una sola tecla o una combinación.
Solo usamos las 2 primeras entradas ya que sabemos que no se pueden usar más de 2 teclas en nuestro fichero de configuración.
Sin embargo orx soporta hasta combinaciones de 4 teclas para una sola entrada.
`orxInput_GetBindingName()` nos brinda una versión de cadena en una entrada como KEY_UP, MOUSE_LEFT o JOY_1 por ejemplo.

PD: Esos son los nombres utilizados en el archivo de configuración para enlazar los botones de las teclas, ratón o joystick para las entradas.

Veamos ahora como manejamos los eventos de FX.

```
if(_pstEvent->eType == orxEVENT_TYPE_FX)
{
  orxFX_EVENT_PAYLOAD *pstPayload;
```

```
  orxOBJECT           *pstObject;

  pstPayload = _pstEvent->pstPayload;
  pstObject  = orxOBJECT(_pstEvent->hRecipient);

  switch(_pstEvent->eID)
  {
    case orxFX_EVENT_START:
      orxLOG("FX <%s>@<%s> has started!", pstPayload->zFXName,
orxObject_GetName(pstObject));

      if(pstObject == pstSoldier)
      {
        // Locks it
        ((MyObject *)orxObject_GetUserData(pstObject))->bLock = orxTRUE;
      }
      break;

    case orxSOUND_EVENT_STOP:
      orxLOG("FX <%s>@<%s> has stoped!", pstPayload->zFXName,
orxObject_GetName(pstObject));

      if(pstObject == pstSoldier)
      {
        // Unlocks it
        ((MyObject *)orxObject_GetUserData(pstObject))->bLock = orxFALSE;
      }
      break;
  }
}
```

Si un FX comienza en nuestro soldado simplemente lo bloqueamos usando nuestra estructura de datos. Reciprocamente, lo desbloqueamos cuando el FX se detiene.

Como hemos cubierto la parte del código, veamos como definimos los FXs mediante código-sabio. Primero echemos un vistazo a un simple FX: la rotación cíclica en la caja.

```
[RotateLoopFX]
SlotList  = Rotate
Loop      = true

[Rotate]
Type        = rotation
StartTime   = 0.0
EndTime     = 2.0
Curve       = sine
Pow         = 2.0
StartValue  = 0
EndValue    = 360

[Box]
```

```
FXList = RotateLoopFX
```

Podemos ver por encima que aplicamos el FX (`RotateLoopFX`) en la caja directamente en su creación y no en el código.
`RotateLoopFX` contiene solo una ranura (`Rotate`), y esta es un bucle (atributo `Loop`).
Definimos entonces la ranura `Rotate`. Todas las veces está expresado en segundos y ángulos en grados.
Hemos definido básicamente una rotación que usa un cuadrado de forma sinusoidal que irá desde 0° a 360° en un periodo de 2 segundos.

Veamos ahora a nuestro FX de oleaje (wobble en Inglés).

```
[WobbleFX]
SlotList = Wobble

[Wobble]
Type          = scale
StartTime     = 0.0
EndTime       = 1.0
Period        = 0.2
Curve         = sine
Amplification = 0.0
StartValue    = (1.0, 1.0, 1.0)
EndValue      = (2.0, 2.0, 1.0) ~ (6.0, 6.0, 1.0)
```

Como puedes ver, estamos ahora modificando la propiedad escala. Dandole un valor inicial(`StartValue`) y un valor final(`EndValue`).
Ambos están expresados en vectores. Pueden haber sido expresados en flotantes si no quisieramos ningún valor anisótropo.
Aunque parezca que estamos usando valores isotrópicos [2], el valor final(`EndValue`) es aleatorio, ej. los componentes X y Y pueden tener valores aleatorios diferentes!
Aparte de esto, la curva usada es una sinusoidal simple durante un periodo de 0.2 segundos, reproducida en un segundo.
Como puedes ver, también usamos una amplificación(`Amplification`) de 0. Esto significa que al final de nuestro tiempo definido(), el factor aplicado será 0, lo que significa que la curva disminuirá con el tiempo para alcanzar finalmente una amplitud de 0 a 1 segundo.
PD: Por defecto, amplificación(`Amplification`) es 1, lo que significa que la curva se mantendrá estable todo el tiempo. Usando un valor menor de 1 disminuirá su ampitud y para un valor mayor a 1 aumentará.

Ahora vamos a echar un vistazo a nuestra propuesta de círculo.

```
[CircleFX]
SlotList        = CircleX#CircleY
KeepInCache     = true

[CircleX]
Type            = position
StartTime       = 0.0
EndTime         = 1.0
Curve           = sine
```

```
StartValue      = (0.0, 0.0, 0.0)
EndValue        = (-50.0, 0.0, 0.0)
UseOrientation  = true
UseScale        = true

[CircleY@CircleX]
Phase       = 0.25
StartValue  = (0.0, -25.0, 0.0)
EndValue    = (0.0, 25.0, 0.0)
```

Aquí necesitamos usar 2 ranuras que afectan la posición así como para ser capaz de tener un movimiento circular.
La primera ranura, `CircleX`, aplicará una curva sinusoidal en el componente X de la posición de nuestro objeto.
La segunda ranura, `CircleY`, aplicará la misma curva(con una amplitud diferente) en su componente Y.

Esto no debería resultar en una movimiento circular si no alteramos la fase(`Phase`) de `CircleY`.
Si tenemos que describir una curva sinusoidal, de fase 0, sería en su valor inicial(`StartValue`), listo para subir la rampa. En fase 0.25, está a punto medio, subiendo la rampa.
En fase 0.5, está en el pico de valor(`EndValue`), cerca de bajar la rampa.
En fase 0.75, está a medio valor nuevamente, bajando la rampa completamente.
En fase 1.0, está exactamente donde mismo la fase 0: valor inicial(`StartValue`), cerca de subir la rampa.
PD: Esta descripción funciona para una curva sinusoidal y también para un único triangulo, pero no para una línea por ejemplo.

Ahora vamos a saltarnos la mayoría de los demás FXs, ya que no tienen nada que no pudieran enterder por ustedes mismos. Sin embargo, echemos un rápido vistazo al volteo, y mostraremos como podemos voltaer un objeto, al estilo de Paper Mario Wii.

```
[FlipFX]
SlotList = Flip

[Flip@Wobble]
EndTime       = 0.5
Period        = 1.0
Amplification = 1.0
EndValue      = (-1.0, 1.0, 1.0)
```

Como puedes ver, simplemente lo logramos con el uso de numeros negativos! 😀
Podemos notar que damos un valor explícito para el periodo(`Period`).
Como se optó por un periodo dos veces más largo que la longitud de nuestra curva definida, significa que solo usaremos la mitad de la curva, ej. solo la parte de subida de la rampa.
También escogimos revertir la amplificación(`Amplification`) de regreso a 1(como el oleaje(Wobble) fue puesto en 0).

Recursos

Código fuente: 07_FX.c

Fichero de configuración: 07_FX.ini

1)
con seno, triangulo, cuadrado o región linear
2)
el componente Z no afecta a los objetos 2D

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

Permanent link:
https://www.orx-project.org/wiki/es/orx/tutorials/fx?rev=1330724570

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